FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Engenharia Electrotécnica

An Agent-Based Evolutionary Approach For Manufacturing System Layout Design

Por
Nuno Emanuel Nunes Pereira

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Electrotécnica e de Computadores

Orientador: Doutor José António Barata de Oliveira

Júri:
Presidente: Doutor Pedro Alexandre da Costa Sousa
Vogal: Doutor José António Barata de Oliveira
Vogal: Doutor João Paulo Branquinho Pimentão

Monte de Caparica
Abril de 2011

An Agent-Based Evolutionary Approach For Manufacturing System Layout Design

Para a minha Família
E os meus Amigos

Acknowledgments

Many people contributed in one way or another to me reaching this point of my life and to the conclusion of this work. They loved me, cared for me and supported me through my life as a student and, more importantly, through my life as a person.

These people are: my mother and father, my grandmother, my brother Bruno Pereira, my cousins Patrícia Nunes and Gonçalo Mateus, my aunts and uncles, Marijke Niessen, Gonçalo Costa, José Belo, José Luzio, Hugo Lopes, Francisco Ganhão, the ”hard core” of the Mestrado Integrado em Engenharia Electrotécnica, and my colleagues and friends outside the university. To all of you, my most sincere thanks.

I want to thank Professor José Barata for the opportunity to develop the work presented in this thesis.

I also thank Dr. Regina Frei for her ideas.

Resumo

Nesta tese é apresentada uma solução para o problema do desenho do *layout* de um sistema de manufactura, uma parte importante da sua fase de projecto, dada a influência que tem na eficiência do sistema e, portanto, na sua capacidade de produção e de lidar com falhas.

A solução apresentada tem por base um algoritmo genético que, utilizando a informação que é fornecida pelo utilizador através de um ficheiro de ontologia, e utilizando algoritmos da teoria de grafos, desenha o *layout* do sistema de manufactura. As instâncias da ontologia representam recursos de manufactura e suas características que, quando utilizados pelo algoritmo, são codificados em cromossomas e nos seus genes.

O algoritmo inicia-se com um número de cromossomas com *fitness* baixa que, com a evolução direccionada proporcionada pelo algoritmo, que é condicionada por parâmetros de controlo que podem ser alterados pelo utilizador, melhoram com o surgimento de novas gerações. Considera-se que a melhor solução é aquela que conecta, por ordem, todos os recursos necessários para realizar as operações requeridas pelo plano de manufactura descrito na ontologia, sem que estes se sobreponham quando o *layout* é construído.

A configuração apresentada pelo sistema de transporte de partes e materiais, no *layout* escolhido, está apenas dependente dos recursos disponíveis e da função de *fitness* utilizada pelo algoritmo genético, sendo que esta última não pode ser alterada pelo utilizador. Esta solução diferencia-se das demais por posicionar simultaneamente todos os componentes do sistema de manufactura e não apenas estações de trabalho ou o sistema de transporte.

A solução é direccionada para *evolvable assembly systems*, pelo que foi implementada dentro de um agente, para poder ser integrada num sistema multiagente, a ser utilizado para controlar um sistema de manufactura, com um mínimo de alterações.

Palavras-chave: desenho de layout, sistema de manufactura, sistema multiagente, ontologia, algoritmo genético.

Abstract

In this thesis it is presented an approach to the problem of layout design for a manufacturing system, which is an important part of its design stage, given that it has influence in the system efficiency and, therefore, in its output rate and fault handling capabilities.

The presented approach is based on a Genetic Algorithm (GA) that, by using information provided by the the user through an ontology file, and by using algorithms from graph-theory, designs the layout of a manufacturing system. The instances of the ontology represent manufacturing resources and their characteristics that, when they are being used by the algorithm, are encoded in chromosomes and in their genes.

The algorithm begins with a number of chromosomes with low fitness which, with the directed evolution provided by the algorithm, that is restricted by the control parameters that might be tunned by the user, improve with the passing of the new generations. It is considered that the fittest solution is the one that connects, in order, all the resources required by the manufacturing plan, described in the ontology, without the occurrence of overlaps when the layout is constructed.

The configuration presented by the transport system that handles parts and materials, in the selected layout, is only dependent on the available resources and on the fitness function used by the GA, being that the last cannot be changed by the user. This approach differs from others by positioning simultaneously all the components of the manufacturing system and not only workstations or transport system.

The solution is directed to evolvable assembly systems, purpose for which it was implemented inside an agent, so it can be integrated in a Multiagent System (MAS) to be used in the control of a manufacturing system with minimal changes.

Keywords: layout design, manufacturing system, multiagent system, ontology, genetic algorithm.

Contents

List of Figures

Acronyms

ACL Agent Communication Language

AEI Advanced Enabling Interface

API Application Programming Interface

AGV Automated Guided Vehicle

BDI Belief-Desire-Intention

CoBASA Coalition Based Approach For Shop Floor Agility

DF Directory Facilitator

EAS Evolvable Assembly System

EUPASS Evolvable Ultra-Precision Assembly Systems

FAS Flexible Assembly System

FIPA Foundation For Intelligent Physical Agents

FMS Flexible Manufacturing System

GA Genetic Algorithm

GUI Graphical User Interface

ICT Information And Communication Technology

IDE Integrated Development Environment

JADE Java Agent Development Framework

Jena Jena Semantic Web Framework For Java

JGAP Java Genetic Algorithms Package

KIF Knowledge Interchange Format

KQML Knowledge Query Manipulation Language

MAS Multiagent System

MGA Messy Genetic Algorithm

NetBeans Netbeans Integrated Development Environment

OS Operating System

OWL Web Ontology Language

Pellet Pellet OWL-DL Reasoner

R&D Research And Development

RDF Resource Description Framework

SA Self-Adaptive

SO Self-Organising

SOA Service-Oriented Architecture

SPARQL SPARQL Query Language For RDF

SWRL Semantic Web Rule Language

URI Uniform Resource Identifier

W3C World Wide Web Consortium

Chapter 1

Introduction

The first humans to inhabit planet Earth were fully dependent on Nature to survive. They would collect from Nature everything they required, not only food and water, but also animal skins and bones, wood and stone which were used to manufacture clothes, shelter and tools [Nunes et al., 1995]. Those times are past but the need to transform what Nature gives in order to satisfy the necessities of society still remains. There are differences though. Nowadays, with the growing understanding of the universe, mankind relies on artificial machines to execute the most difficult, repetitive and precision requiring tasks, which can be witnessed in different domains such as search and rescue, medicine, warfare, space exploration, construction and manufacturing [Frei et al., 2009b].

1.1 Context

The purpose of a manufacturing company is to manufacture a product that will enter the market where it will be available for consumers to buy. Consumers will only purchase what they need, like and can afford. In this way, they have great influence in the manufacturing world. Furthermore, unless a company has the monopoly[1] of the market, competing companies are always a concern, since they will try to make their products more appealing to consumers then those of other companies. For both these reasons, a manufacturing company has to manufacture products that are attractive to consumers in terms of quality and cost, and that often have to be improved or renewed given their short life-cycle.

1.2 Problem

Companies have to be able to cope with the market demands described in Section 1.1 while maintaining the manufacturing costs of their products as low as possible. Tompkins et al. [2010] stated that between 20 and 50% of the total operating expenses within manufacturing is attributed to material handling. The

[1] A monopoly is an exclusive ownership through legal privilege, command of supply, or concerted action ['monopoly', 2010]

efficiency of a material handling system is dependent on the layout of the whole manufacturing system, which is difficult to design and costly to modify [Rajasekharan et al., 1998]. The problem of generating the best layout for a manufacturing system is very old and, due to the complexity of designing a system flexible to handle a variety of product requirements and agile to handle the introduction of new products, research efforts have only produced a number of incomplete solutions.

1.3 Hypothesis

A manufacturing system capable of self-organising provides a way of addressing the problem described in Section 1.2 by automating part of the layout design process. The following hypothesis are formulated: a manufacturing system is capable of designing its layout using self-organisation if it has access to enough information about itself and the world around it. It is possible to define and implement an architecture able to create its own layout. The resulting implementation facilitates the the task of designing a manufacturing system's layout.

1.4 Objectives And Contributions

The objective and contribution of this thesis is to design and implement an agent architecture compliant with the hypothesis of Section 1.3. This architecture is capable of designing the layout of a manufacturing system by integrating agent technology, ontologies, genetic algorithms and graph-theory algorithms.

1.5 Structure of the Dissertation

The dissertation is structured in eight chapters:

- Chapter 2 - state of the art of manufacturing technology and related work: manufacturing paradigms, control approaches, agent-based control, ontologies, genetic algorithms, layout design, and graph-theory algorithms.
- Chapter 3 - agent's architecture and integration of modules: ontologies, genetic algorithms and graph-theory algorithms.
- Chapter 4 - support ontology that describes the manufacturing plan and manufacturing resources.
- Chapter 5 - genetic algorithm: coding of the chromosomes, mutation operators and fitness functions.
- Chapter 6 - implementation: problems encountered and solutions found.

- Chapter 7 - tests performed to validate the implementation.
- Chapter 8 - analyses the work developed in this dissertation, based on the hypothesis of section 1.3, and future work.

Chapter 2

State Of The Art, Related Work And Supporting Concepts

A manufacturing company integrates product development and design activities with the process of producing the product and with business, which involves distribution, marketing and service infrastructure [Leitão, 2004]. The work developed in this thesis focuses mainly in the product description and corresponding manufacturing system.

2.1 Manufacturing Paradigms

A manufacturing system is a combination of machines, tools and human workers organised in workstations and interconnected by transporters and material handling technology. The manufacturing process results of the coordinated action of humans and manufacturing equipment and is defined by Groover [2007]:

> "Manufacturing can be defined as the application of physical and chemical processes to alter the geometry, properties, and/or appearance of a given starting material to make parts or products; manufacturing also includes the joining of multiple parts to make assembled products."

This means that product, process and system are intrinsically related to one another. Hence, a change in product design may have an impact on the process and the system. Likewise, a change in a process may imply a change in the product design and system [Frei et al., 2009b]. Nonetheless, a manufacturing system should be able to respond instantaneously to product demands [Davidow and Malone, 1993]. However, the readiness and quality of that response is dependent on the control architecture and physical layout of the system. Although the majority of manufacturing companies remains resilient to the replacement of their antiquated but well known manufacturing system of many years by a strange and complex

system that just came out of Research And Development (R&D), manufacturing paradigms have been evolving through the years and the ones available today allow the design of flexible and agile systems that are able to deal with product diversity, change and uncertainty.

2.1.1 Flexible Manufacturing Systems

Flexibility is the capability to adapt to new, different, or changing requirements ['flexibility', 2010]. A manufacturing system gains this capability when it is designed to handle predictable variations. Leitão [2004] identifies different types of flexibility, such as mix, changeover, volume, product and sequencing. Each type is characterized by dealing with a different variation:

- mix flexibility - the capability to deal with a range of products or variants;
- changeover flexibility - the capability to quickly change the manufacturing system to offer a new product;
- volume flexibility - the capability to deal with demands of variable volume;
- product flexibility - the capability to quickly modify a product design;
- sequencing flexibility - the capability to support different sequences of the operations that are part of the manufacturing plan.

The control of a Flexible Manufacturing System (FMS) is carried out by an integrated computational system [Upton, 1992] and its main limitation is its inflexibility to the introduction of new products due to the complexity of automatically making the required adjustments [Leitão, 2004].

2.1.2 Intelligent Supervising Systems

Intelligent Supervising Systems are an approach to FMS [Camarinha-Matos et al., 1996]. They include a planning module which is responsible for the generation of executable manufacturing plans. However, the generated plan is based on a simplified model of the world, which does not consider every hypothesis and, even if it did, there can be the occurrence of unpredictable events. Therefore, there is the need for a supervising module, which is responsible for executing the plan and generating eventual recovery plans. The supervising module is composed of:

- dispatcher - Executes the manufacturing plan, given by the planning module, by sending execution commands to the device controllers.
- monitoring - Monitors the manufacturing process, based on the information retrieved from sensors, in search of deviations from the plan objectives, problems or abnormalities.

- diagnosis - Analyses the issues encountered and identifies their origin.
- recovery - Tries to recover from the issue by generating a recovery manufacturing plan.

Figure 2.1 shows the information flow between these modules in a possible supervising architecture.

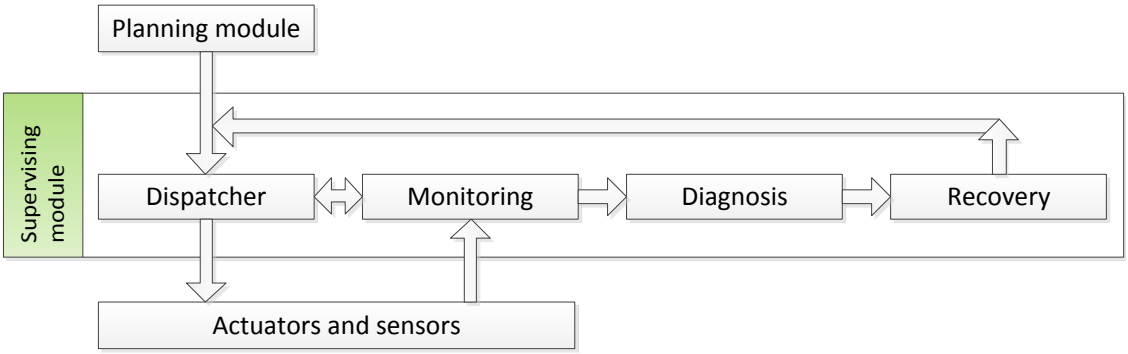

Figure 2.1: Information flow in a supervising architecture

Supervising applications may include extra functionalities:

- prognosis - Analyses the temporal evolution of a set of state variables and tries to anticipate the occurrence of errors.
- preventive maintenance - Similarly to recovery, this functionality generates preventive maintenance plans.
- learning - Improves the performance of the previous functionalities through the acquisition of knowledge.

Supervising systems are applied when a real-time system is required to handle different asynchronous events in a certain time limit. If expert techniques from the artificial intelligence domain are used in the implementation of a real-time system, a real-time expert system is obtained.

Expert Systems

Winston and Prendergast [1984] define an expert system as ”a computer program that behaves like a human expert in some useful ways”. They involve modeling knowledge through logical propositions and reasoning with it through the use of rules. The system core architecture is composed of a knowledge base and an inference engine but an user interface and a data base are also required to interact with the user and store relevant data, respectively.

2.1.3 Agility in Manufacturing Systems

Agility is the ability to thrive and prosper in an environment of constant and unpredictable change [Goldman et al., 1995] and it covers different areas of manufacturing, from management to shop floor [Barata, 2005]. A manufacturing system that has this ability is able to deal with change and uncertainty [Barata, 2005] and with things that cannot be controlled [Maskell, 2001]. Barata [2005] includes a list of requirements for a successful implementation of agile manufacturing, from which the following points are relevant in the context of this thesis:

- information technology - Information And Communication Technologies (ICTs) are required to provide the computational support needed for the creation of an agile shop floor.
- new agile shop floor strategies - Actors, processes and areas involved in the manufacturing system should be studied for the development of a methodology that helps the integration of these elements during the creation of an agile shop floor to support their life cycle evolution.

2.1.4 Evolvable Assembly Systems

Evolvable Assembly Systems (EASs) arose as a solution to the problem of achieving agility in assembly systems. They are based on the concept of emergence whose basic idea is that complex systems exhibit expected and unexpected properties and behaviours resulting from the integration of components with known and unknown characteristics. Their composition consists of several reconfigurable, task-specific and process oriented modules which allows for a continuous evolution together with the product and the corresponding assembly process. These characteristics allow modules to live beyond the product life cycle [Frei et al., 2009a, Barata and Onori, 2006, Onori et al., 2005].

Barata and Onori [2006] list some requirements to successfully designing an EAS:

- module - Any unit that can perform operations and integrates a specific interface.
- granularity - The lowest level of device being considered in the reference architecture. The lower the level of the building blocks the higher the emergent behaviour.
- plugability - The ability to rearrange and integrate system components within the framework of a given system architecture. A legacy system[1] can be adapted into an EAS component through an Advanced Enabling Interface (AEI) that enables the communication with other EAS components.
- reconfigurability (interoperability) - The ability to rearrange available system components to perform new, but predefined functions.

[1] A legacy system is an antiquated computer system which is still in use because it cannot be replaced or redesigned.

- evolvability - The reconfiguration of the system platform enables new or refined levels of functionality.

The architecture of an EAS is composed of an individual module architecture and a global system architecture.

Individual Module

An individual physical module is represented by a computational model that must be part of the architecture. This model must possess a set of attributes that capture the relevant physical characteristics of the module and are to be used for its operation, configuration and selection. Model functionalities should capture the behaviour of the physical module and realize the necessary control actions that must be issued for the behaviour to be accomplished [Onori et al., 2005]. They are offered as skills and each skill represents the capability of the module to perform a certain task, which may be dependent on some or all the represented attributes. A skill execution may involve performing a sequence of the control actions offered by the module controller. A control action should be offered in a skill execution sequence, and not alone, when its resulting effect is not significantly relevant to the assembly process.

Assembly System

The global architecture of the assembly system should support easy addition and removal of modules and also support their interactions, such as synchronization, which are required to handle complex skills (see Figure 2.2 (equipment figures taken from [KUKA, 2010, ABB, 2011, Montech, 2011, SCHUNK, 2011])). A complex skill is a skill offered and performed by a group of modules selected by the user to be part of a coalition [Barata, 2005, Onori et al., 2005].

Development Of EAS

The Coalition Based Approach For Shop Floor Agility (CoBASA) is a multiagent approach developed by Barata [2005] which established the first ideas and concepts for EASs [Frei et al., 2008].

The Evolvable Ultra-Precision Assembly Systems (EUPASS) Integrated project, which ran from 2004 to 2009, aimed to develop affordable, cost effective and sustainable ultra-precision manufacturing solutions by offering rapidly deployable ultra-precision assembly services on demand, which was to be achieved by developing and delivering a number of breakthrough technologies and solutions [EUPASS]. This endeavour followed the EAS principles, and has, together with more pinpointed efforts between KTH, UNINOVA, and EPFL, elaborated a set of foundations for EAS [Shen et al., 2006].

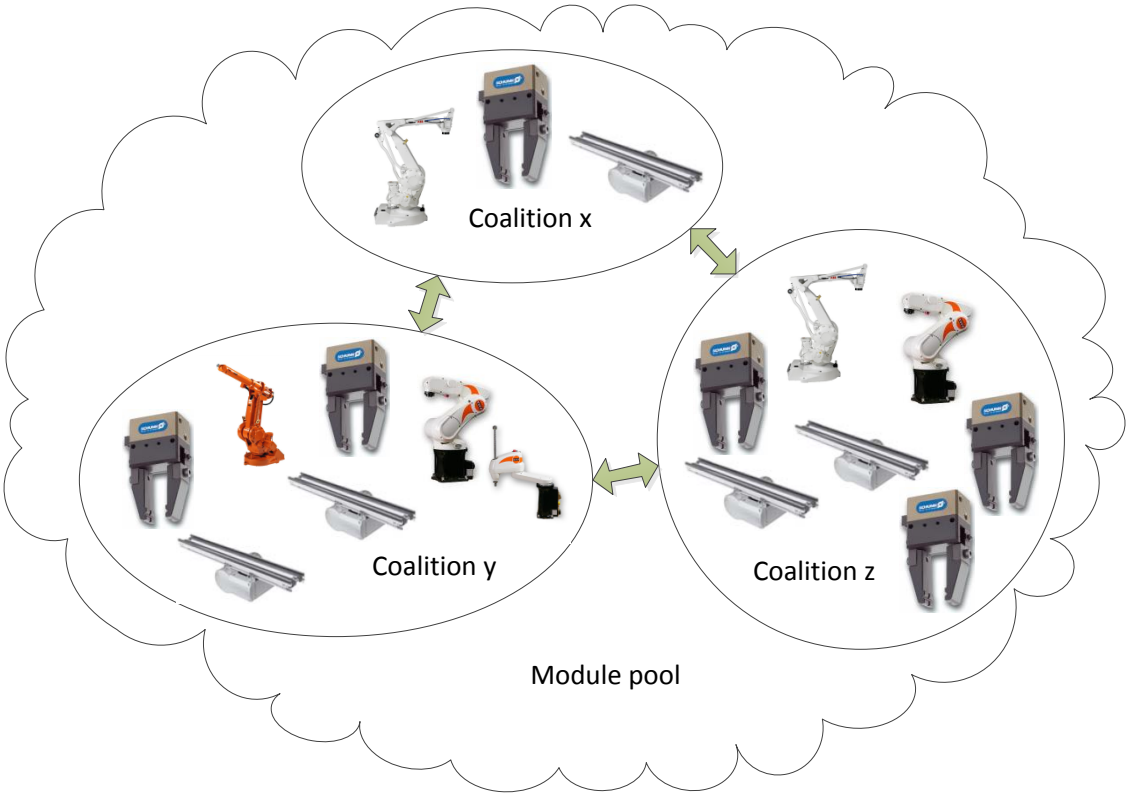

Figure 2.2: An evolvable assembly system architecture

2.1.5 Self-Organisation in Manufacturing Systems

Although manufacturing systems such as EASs are built to be easily reconfigurable, a human is still required to actually design the reconfigured system, to program the required modules, and to monitor the manufacturing process. The whole process is expensive, error-prone and tedious [Frei et al., 2009a]. The application of self-* capabilities to manufacturing systems contributes to the creation of more user-friendly systems by increasing autonomy and hiding complexity from the user [Frei and Barata, 2008]. However, to obtain a trustworthy system, the user retains control, and safety, security and performance must be assured during development, deployment and evolution [Di Marzo Serugendo et al., 2008]. Babaoglu and Shrobe [2010] define Self-Organising (SO) systems and Self-Adaptive (SA) systems:

> ”Self-organising systems work bottom up. They are composed of a large number of components that interact locally according to typically simple rules. The global behavior of the system emerges from these local interactions. Here, a challenge is often to predict and control the resulting global behavior.”

> ”Self-adaptive systems work in a top down manner. They evaluate their own global behavior and change it when the evaluation indicates that they are not accomplishing what they

were intended to do, or when better functionality or performance is possible. A challenge is often to identify how to change specific behaviors to achieve the desired improvement."

Trustworthy SO and SA systems can be created by assuring their compliance with the following requirements, which were adapted from Di Marzo Serugendo et al. [2008]:

- autonomous individual components - Robustness and self-* behaviour arise from the system being composed of autonomous individual components, which can be ant-like entities in an SO system and autonomous parts of the supporting infrastructure in an SA system.
- interoperability - The global behaviour of the system arises from the interactions between the individual components of the SO or SA system. Description of component capabilities should be decoupled from the programming code.
- self-awareness - Self-* properties and behaviour arise from the capability of the system or its individual components to identify by themselves new opportunities of evolution, problems and solutions, without the interference of a human user. Self-awareness involves sensing capabilities in order to trigger reasoning and acting, and requires acquisition, updating and monitoring of metadata[2]. The components of an SO system are capable of sensing and affecting their environment. SA systems use intelligent supervising components, such as planners, controllers and monitors.
- behaviour guiding and bounding - A set of rules are used to direct and limit the behaviour of SO and SA system components towards what might me considered an optimum. In a SA system, there are also rules that apply to the system as a whole. The system must possess mechanisms for enforcing these rules.
- development process - The development process involves the analysis of the system in different views from abstract descriptions to programming code. Self-description of components and specification of metadata and rules must be available at design-time, run-time or both, depending on when they are required.

The main difference between a SO system and a SA system is that the first is decentralised and bottom up driven and the second is hierarchical and top down driven.

Self-Organising Evolvable Assembly Systems

A SO-EAS is an EAS in which the modules self-organise to create a suitable layout for the assembly and that self-adapts as a whole to manufacturing conditions [Frei et al., 2008]. The static coalitions of

[2]Metadata conveys functional and non-functional information about the system itself [Di Marzo Serugendo et al., 2008]

classic EASs become dynamic. As an example of application, when new modules are connected to an EAS, their controllers communicate with one another, verify their compatibility, create complex skills based on their individual simple skills, and offer them to the user [Frei and Barata, 2008].

Frei et al. [2009b, 2010], Frei [2010] investigate the question of whether self-organisation can be useful in agile assembly, what are the suitable mechanisms for self-organisation, and how they can be implemented. The work developed in this thesis is part of this research.

2.2 Control Approaches

Shop floor control approaches have been evolving through time with a noticed increasing of autonomy and relaxation of master-slave relationships [Barata, 2005], which allowed the arising of more flexible and agile manufacturing systems. Dilts et al. [1991] classifies control approaches as centralized, hierarchical and heterarchical, which have advantages and disadvantages as Barata [2005], Leitão [2004] have identified.

2.2.1 Centralised

In a centralised approach (see Figure 2.3), the central module is responsible for all decision-making related with planning and processing of information and issues commands to the modules around it. By requiring simpler coordinating algorithms, it eases management and control optimization. However, introducing modifications and adding new elements implies making changes to the program. Moreover, its total dependence on the central note, raises control complexity and response time, and lowers its fault tolerance.

2.2.2 Hierarchical

In this approach, decision-making is divided by multiple modules organized hierarchically in a tree-like structure (see Figure 2.4). Modules in upper layers of the tree interact with those on a layer immediately below based on the master-slave concept. Hence, the module on the top of the hierarchy sets the global goals and the long-range strategies and issues commands to the lower levels which decompose these commands into simpler ones and then send them to a further lower level for their accomplishment. Thus, the complexity of each individual controller is reduced. A system implementing this architecture works near optimal performance under stable situations but the appearance of disturbances reduces that performance significantly. As a response to disturbances, there can be a control loop for adaptive purposes, which is possible because information also flows bottom-up. Unlike a centralised approach, this approach enables modifications such as the addition of new modules to existing layers. However, it is

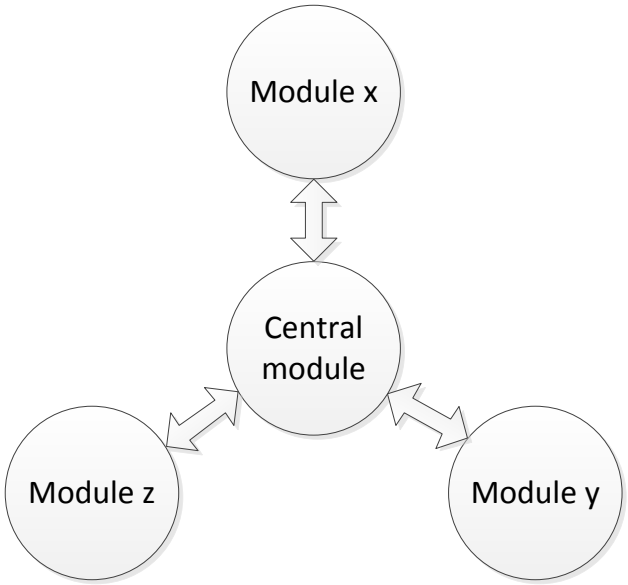

Figure 2.3: Interactions in a centralised control architecture

difficult to introduce a new layer.

2.2.3 Modified Hierarchical

The modified hierarchical approach maintains all features of the hierarchical approach while adding interaction between modules at the same hierarchical level (see Figure 2.5), increasing their complexity. In this way, whenever there is a disturbance in the normal execution of the system, it might be solved with the information gathered at the level where it was detected and without involving upper levels, improving response. System expandability is also improved.

2.2.4 Heterarchical

This is a flat architecture (see Figure 2.6) where decision-making is distributed and happens in each module autonomously, without a global view of the system. Thus, overall complexity is reduced when compared with other approaches. Modules cooperate with one another and it is easy to modify their functioning and to add new modules to the system, which makes expandability an easier task than in other approaches. It increases performance against disturbances by decreasing response time at the cost of reducing global optimisation.

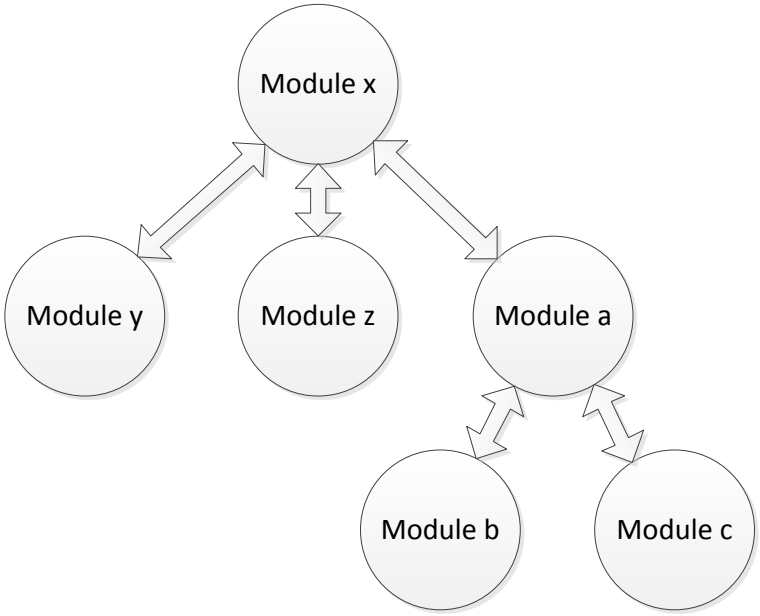

Figure 2.4: Interactions in a hierarchical control architecture

2.3 Agent-Based Control

The heterarchical architecture, referred to in 2.2.4, is also designated by autonomous agent approach in the agent domain [Leitão, 2004]. In this approach, the manufacturing system is controlled by a society of autonomous entities called agents which interact autonomously with the environment through the devices they control, and with one another through the exchange of messages. There are two levels of abstraction to be considered: the individual agent and the multiagent society.

2.3.1 Individual Agent

Wooldridge [2009] gives the following definition of agent:

> ”An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives.”

This definition of agent was chosen because it is in agreement with the requirements of this thesis but it does not represent a universally accepted definition of the term. Such definition does not exist, which is due to the fact that the various attributes associated with agency are of differing importance for different domains, from which autonomy is considered central to the notion of agency [Wooldridge, 2009]. Therefore, an agent presents a desired level of autonomous behaviour in its interactions with the environment and with other agents. This autonomy is dependent on the decisions it is prepared to make, based on an internal representation of the perceived environment, which can be thought of as an internal state, in

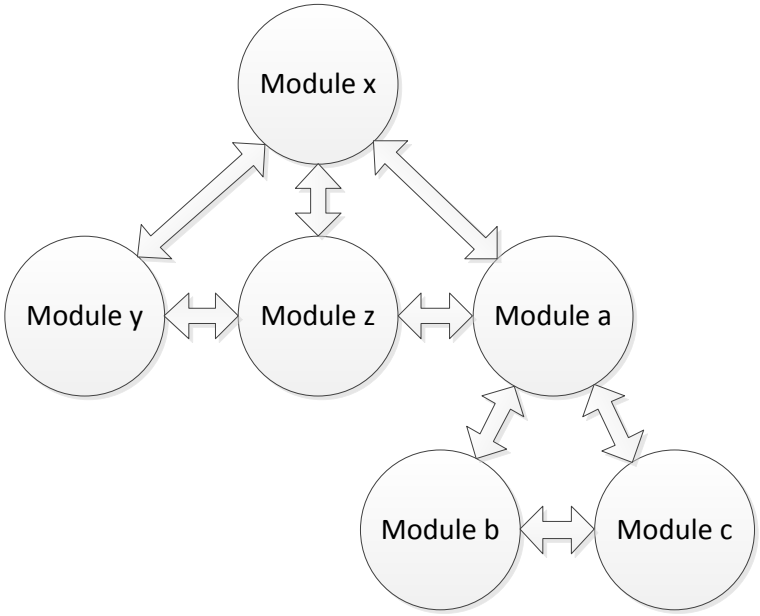

Figure 2.5: Interactions in a modified hierarchical control architecture

order to take action towards the realization of its functionality. Perception, decision and action are the keywords of agent behaviour.

2.3.2 Agent Typologies

An agent can be classified according to its behaviour as deliberative, reactive and hybrid [Wooldridge, 2009].

Deliberative Agent

A deliberative agent is an agent that behaves pro-actively towards the achievement of a predetermined goal. It maintains a knowledge representation of the world and, through reasoning, it is capable of planning a course of action, which comprehends the generation of a correct and optimal sequence of actions that take him closer to its goal [Leitão, 2004].

The Belief-Desire-Intention (BDI) architecture is a well known deliberative agent architecture, which attempts to replicate human practical reasoning [Wooldridge, 2009]. In this architecture, the reasoning process that leads to decision-making takes in consideration the following representations:

- beliefs - Knowledge of the agent regarding itself and its environment.
- desires - Goals that the agent has to achieve, but does not know yet how to.
- intentions - Goals that the agent is committed to achieve and that knows how to.

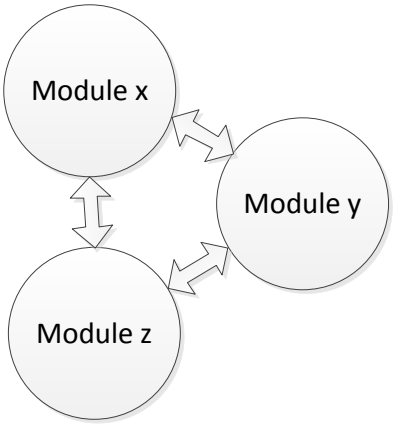

Figure 2.6: Interactions in a heterarchical control architecture

Reactive Agent

A reactive agent is an agent whose sole purpose is to respond to changes that occur in the environment, when they happen, producing robust actions. Unlike a deliberative agent, it does not have an internal knowledge representation.

Brooks [1986, 1991a,b,c] proposed the subsumption architecture for reactive agents, which is based on three thesis:

- Intelligent behaviour does not require explicit symbolic representations.
- Intelligent behaviour does not require explicit abstract symbolic reasoning.
- Intelligence is an emergent property of certain complex systems.

The basic ideas are that intelligence is in the world and not inside the agent. Moreover, intelligent behaviour arises from the interaction of the agent with its environment.

This architecture is a hierarchy of behaviours and each behaviour has a situation-action rule-like structure. Although behaviours compete with one another for the control of the agent, lower layers of the architecture, which represent more primitive kinds of behaviour, have precedence over layers further up the hierarchy Wooldridge [2009].

Hybrid Agent

A hybrid agent combines the best characteristics of deliberative agents and reactive agents. Their architecture is composed of a set of interacting layers in which some are deliberative and others are reactive. Therefore, they are capable of generating optimal sequences of actions and of achieving fast response.

However, depending on how the layers receive sensorial information and interact, this architecture can be classified as horizontal layering and vertical layering [Wooldridge, 2009].

In horizontal layering, the sensors are directly connected to the layers, which are directly connected to the output [Wooldridge, 2009]. This can create a problematic situation in which different layers give different orders to the output. [Barata, 2005].

In vertical layering, sensors and output connect to one layer each, which can happen in one of two ways. In ”one pass control” sensors are connected to the bottom layer and actuators are connected to the top layer; consequently, required information flows bottom-up, being filtered from layer to layer. In ”two pass control” sensors and actuators are connected to the bottom layer. Hence, information flows upwards and then downwards through the layers [Wooldridge, 2009]. In both situations, only one layer affects the output. Thus, the problem described in horizontal layering does not occur in this architecture. However, if one layer stops working, the whole architecture fails [Barata, 2005].

2.3.3 Multiagent Systems

Agents in a Multiagent System (MAS) have to interact with one-another. Wooldridge [2009] states that this interaction requires the abilities:

- coordination - Coordination is needed in the access to non-sharable resources.
- cooperation - Agents cooperate to achieve a common goal by working together.
- negotiation - The process of negotiation is required to reach agreements on matters of common interest.

The usefulness of these abilities is observable in a coalition, which is a structure that results from the association of at least two cooperating agents. Negotiation is required for the autonomous formation of coalitions, and coordination is required when performing a joint activity.

Coalition Formation

Sandholm et al. [1999] identified three activities in coalition formation:

- Coalition structure generation: the set of agents should be partitioned in coalitions in such a way that the agents inside a coalition coordinate their efforts with one-another but not with agents belonging to other coalitions.
- Solving the optimization problem of each coalition. This involves coordinating the individual efforts and the usage of resources of each agent while trying to maximize the efficiency of the resulting joint effort.

- Dividing the value of the generated solution among agents.

2.3.4 Communication Technologies

Agent interaction involves communication and understanding which is where ontologies (see 2.4), Agent Communication Languages (ACLs) and agent communication protocols play an important role.

Agent Communication Languages

ACLs provide formats for the exchange of messages between the agents in the MAS. There are two well known ACLs: Knowledge Query Manipulation Language (KQML), developed by the ARPA knowledge sharing initiative, and FIPA-ACL, developed by Foundation For Intelligent Physical Agents (FIPA). The structure of a message is similar in both standards, being composed of a performative, a content and a set of control parameters [FIPA, 1996 - 2002b]. Performative and content parameters are the two components of a speech act, as explained in speech act theory [Austin, 1975, Searle, 1997], and are used by agents to express their intentions to other agents. Some FIPA-ACL performatives are relevant to this work, such as: request, agree, refuse, inform, failure, cfp (call for proposals), propose, accept-proposal, reject-proposal and subscribe. KQML includes the Knowledge Interchange Format (KIF) language, which is used to express the content of messages.

Agent Communication Protocols

A communication protocol defines how agents should behave when communicating with one-another. They consist of an ordered sequence of messages that an agent should send, or wait for, at a given time during the communication process. FIPA defined a set of communication protocols, some of which are relevant to this work, such as: request interaction protocol, contract net interaction protocol, subscribe interaction protocol [FIPA, 1996 - 2002c,-,-]. The performative of a message sent or received is related with the protocol in use at that time and with its state of execution.

2.3.5 Integrating Agents In Custom Applications: The Jade Framework

Java Agent Development Framework (JADE) [Jade - Telecom Italia Lab, 2010] is an open source platform developed by Telecom Italia for peer-to-peer agent based applications fully implemented in the Java language. It can be used to implement multi-agent systems through middle-ware that is compliant with the FIPA specifications, and through a set of graphical tools that facilitate debugging and deployment phases.

2.4 Ontologies

An ontology is a formal specification of a set of terms used to describe and represent an area of knowledge. It is widely used in areas such as knowledge engineering, artificial intelligence and computer science in computer applications that involve knowledge management and information management [Barata and Onori, 2006]. In the context of this work, an ontology is used to define a set of classes of things, the properties of those classes and the relationships between them. Moreover, the classes and properties defined are instantiated to create a set of individuals which represent a knowledge base of the corresponding ontology domain.

Technologies available for developing ontologies include language specifications and editing tools. As for integrating ontologies in an application, there are libraries which can be imported into a programming environment and that make it easy to read and write ontology files.

2.4.1 Language Specifications: RDF and OWL

The two languages that are relevant to this work are Resource Description Framework (RDF) and Web Ontology Language (OWL), both developed by the World Wide Web Consortium (W3C). These languages use syntax in the encoding of knowledge.

RDF

RDF is a standard for data interchange. In this language, a thing is represented as a resource which is identified by a Uniform Resource Identifier (URI). It encodes information in triple structures, which are composed of three URIs: URI of a resource, followed by the URI of a property, followed by a literal or the URI of other resource. Properties can have a literal or a URI as their value. This linking structure forms a directed labeled graph where the nodes represent resources and edges represent properties. This facilitates visualization and analysis of the ontology [RDF Working Group, 2009].

OWL

OWL is useful in applications where information is to be processed as it is capable of describing classes of concepts with the corresponding properties and relations between them. Moreover, classes have restrictions and properties have characteristics and constraints. There are three increasingly-expressive sublanguages of OWL: OWL Lite, OWL DL and OWL Full, which differ from one-another in the relations and restrictions allowed on classes and in the characteristics and constraints allowed on properties. This language allows the instantiation of classes to create individuals and it is the decision of the creator of an ontology to decide if a thing is a class or an individual. As resources in RDF, classes and individuals in OWL have an URI by which they are identified. Similarly, the value of a property can be an URI

or a datatype, such as: boolean, integer, double, string. The OWL Working Group has produced a W3C recommendation for OWL [OWL Working Group, 2004].

2.4.2 Editing Tools: Protégé

The editing tool used in the context of this work was Protégé, a free open source ontology editor and knowledge-base framework which supports OWL [Stanford Center for Biomedical Informatics Research, 2010]. The creator of an ontology can use the Graphical User Interface (GUI) of Protégé to write his ontology and then use the automatically generated files in a computer application or program. This editor supports the addition of third party plug-ins, which add new functionalities to the ones available by default, facilitating the process of creating an ontology.

2.4.3 Integrating Ontologies In Custom Applications: Jena Java API

Jena Semantic Web Framework For Java (Jena) [Jena] is a Java Application Programming Interface (API) which supplies methods for creation and manipulation of RDF graphs [McBride et al., 2009]. In Jena, graphs, resources, properties and literals are represented by Java object classes and interfaces. Moreover, a graph is called a model, an arc of a model is called a statement and each statement asserts a fact about a resource, which is accomplished by its three components: subject, predicate and object. These components correspond to the three components of a triple.

Jena also aims to provide a consistent programming interface for the development of applications which deal with ontologies [Dickinson, 2009]. One of the ontology languages it supports is OWL. However, the Jena ontology API is used equally for every language supported. It provides Java object classes for ontology classes, object properties, datatype properties and individuals. The value of an object property is an URI and the value of a datatype property is a datatype. The ontology is contained in an ontology model, which is an extension of the model used for RDF, as the information is still encoded in RDF triples. It also facilitates working with ontologies spread across different files as imports through a document manager.

2.4.4 Manufacturing Ontologies: EUPASS Ontology

A manufacturing ontology is an ontology that defines concepts in the domain of manufacturing, such as manufacturing system components and processes. An example of manufacturing ontology is the ontology produced in the EUPASS project, which is a relevant reference to the work developed in this thesis.

2.5 Genetic Algorithms

Genetic Algorithms (GAs) [Goldberg, 1989] are inspired by natural genetic evolution, the process by which nature produces new individuals different from their parents and that leads to the arising of new species. In this process, favourable individual differences and variations are preserved, and those that are injurious are destroyed, which is known as natural selection or survival of the fittest [Darwin, 2010]. In nature, genetic information of individuals is encoded in structures called chromosomes. When two individuals reproduce, their genetic information is combined to create new chromosomes, which contain genetic information used to generate a new individual.

A genetic algorithm is a simplified version of this process that uses data structures as its chromosomes. In such an algorithm, a population of individuals undergoes a cycle of evolution (see Figure 2.7). Each iteration of this cycle comprises: selection of parent chromosomes, their duplication and crossover of the genes, mutation of the resulting offspring, and selection of the fittest who will remain for the next iteration. Its purpose is to search for an optimal solution to a given problem, which might not be the best. Heuristics can be used to fasten the search.

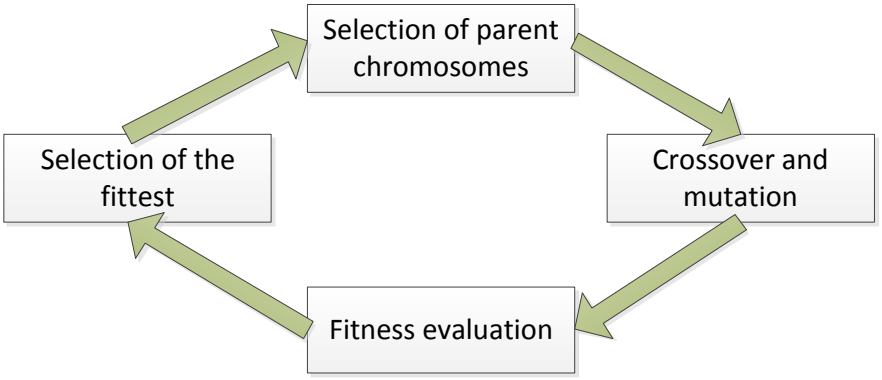

Figure 2.7: A genetic algorithm evolution cycle

2.5.1 Encoding Information

Information of individuals is encoded in a data structures called chromosomes, which are composed of genes (see Figure 2.8). A chromosome can be structured as a sequence of genes or can have multiple layers. Each individual is a possible solution to the problem, and, by having different chromosomes, each individual represents a different solution. An allele, which is the value of a gene, can be a number, letter, symbol, or a complex data structure. In spite of all chromosomes having the same length in classical GAs, in messy GAs, it is possible for different chromosomes to have different lengths.

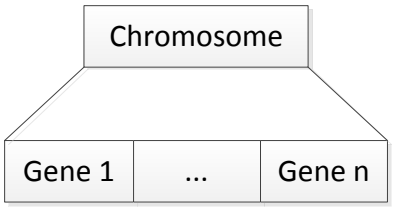

Figure 2.8: The structure of a chromosome

2.5.2 Selection Of Parents

The process of reproduction requires the selection of multiple pairs of individuals to be the parents of new individuals, which will join the population for the following iteration. The most common algorithm used for this step is the roulette wheel, where each individual in the population is given a probability of being selected proportional to its fitness, which is used for randomly choosing pairs of parent chromosomes.

2.5.3 Crossover Operator

Crossover is the process by which the copies of two parents exchange sequences of genes from their chromosomes, in order to create two new individuals. Single point crossover, which is the most common type of crossover, consists of choosing a position in the gene sequence of one parent chromosome, being that such a position is designated as locus, and exchanging all genes from that point on to the other chromosome and vice-versa. Occurrence of crossover is dependent on a given probability. If it does not happen, the two chosen parent chromosomes are copied directly into the population.

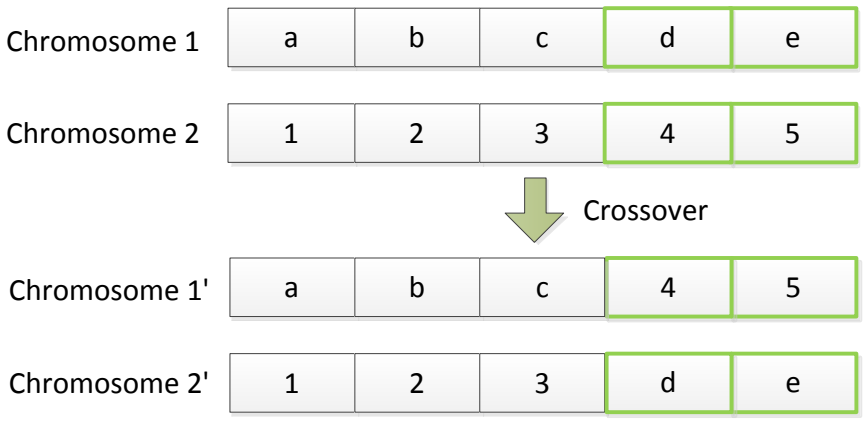

Figure 2.9: Single point crossover effect

2.5.4 Mutation Operator

Mutation is the process by which the genes in a chromosome change their alleles with a given probability. This process consists of choosing a gene of the chromosome at random chance and changing its value to other possible value.

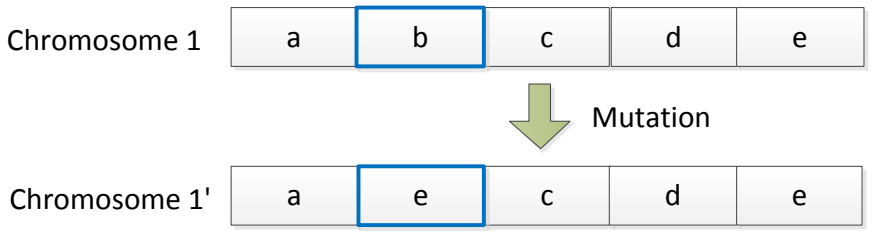

Figure 2.10: Mutation effect

2.5.5 Fitness Evaluation

Only the fittest chromosomes of an iteration are kept to the following one. For this purpose, each chromosome is evaluated and given a fitness value. The ones which have the higher value are the ones that are kept. Evaluation is performed by an evaluation function, which returns a fitness value. This function requires the decoding of the information encoded in the chromosomes. Decoding is the process of retrieving the real value of the information represented by the genes in a chromosome.

2.5.6 Integrating Genetic Algorithms In Custom Applications: JGAP

Java Genetic Algorithms Package (JGAP) [Meffert and Rotstan, 2002 - 2010] is a framework for genetic algorithms and genetic programming for the Java language. It provides basic genetic mechanisms that are useful in the application of evolutionary principles to problems that are to be computationally solved. It contains classes for chromosomes, genes, genetic operators and fitness functions that can be used or extended as it is necessary.

2.6 Manufacturing System Design

The life cycle of a manufacturing system can be summarized in a design time stage followed by a run-time stage. In design time, the system is being designed and implemented by a software architect in order to be fully functional to enter the run-time stage, when it will be executing and manufacturing the product. Manufacturing solutions such as the previously discussed intelligent supervising systems, EASs

and SO-systems increase system complexity and introduce new stages in its life cycle. SO principles introduce creation-time, a stage when the layout of the system is being generated by software. Intelligent supervising systems and SA principles introduce stages when the manufacturing plan is improved, modified or replaced by other, and the system is reconfigured. EAS introduces the possibility for a system to evolve. This thesis addresses design time and creation-time issues, such as the choosing of the required equipment modules and the design of the shop floor layout.

The process of designing a manufacturing system consists of selecting production and material handling equipment, and determining a layout for this equipment, which are different problems on their own [Heragu and Kusiak, 1988]. Depending on the situation, manufacturing and material handling equipment can be selected and laid out at the same time, or manufacturing equipment can be first, followed by material handling equipment. The usage of one or other approach is dependent on each situation analysis [Heragu, 2006].

2.6.1 Equipment Selection

A designer must know what are the required types and quantities of manufacturing and support equipment before designing the machine layout. The required types of equipment can be determined by matching available equipment with the basic production processes required on product parts and materials to produce the final product [Heragu, 2006]. As for quantities of equipment, which is to be understood as the number of pieces of each type of equipment, the designer must account for sequencing of operations and fault tolerance, which are relevant issues during run-time and are affected by this decision. If the right types and quantities of equipment are selected, then the costs of equipment purchase, operation and maintenance will be efficient, machine utilization will be increased, and available space will be efficiently used [Heragu and Kusiak, 1987].

Tompkins et al. [2010] describes material handling equipment selection as a problem of ”providing the right amount of the right material, in the right condition, at the right place, at the right time, in the right position, in the right sequence, and for the right cost, using the right methods”. Properties and characteristics of materials, such as: size, weight, form, etc, must be considered when selecting material handling equipment. Heragu [2006] identifies several types of material handling devices: conveyors, palletizers, pallet lifting devices, trucks, robots, Automated Guided Vehicles (AGVs), hoists, cranes and jibs, and warehouse dedicated devices.

2.6.2 Equipment Layout

The position and orientation of each machine in the shop floor and the way materials are handled and transported from one workstation to the other, affects the efficiency of a manufacturing system. For an

efficient layout design, the designer must consider: throughput rates, responsiveness, material handling efficiency, scope, scale, equipment and housing costs, reconfigurability, product life cycle, fault tolerance, interactions with the environment, etc. A flexible, modular and reconfigurable layout might not need to be redesigned each time manufacturing requirements change. Relayout can be expensive and might require shutting down production [Benjaafar et al., 2002].

2.6.3 Layout Configurations

Depending on the situation, different types of layout can be used in the design of a manufacturing system. Product, process and cellular layouts are designed for a specific product mix and production volume that are supposed to last for a long period of time [Heragu, 2006].

Transfer/Flow Line Layout

In a transfer/flow line layout, workstations are layout in line, ordered according to the required sequence of operations. Consequently, during run-time, materials flow from one workstation to the next in a sequential manner. The execution of one operation is dependent on the finalization of the previous one, which means that the production rate is dependent on the slowest operation. The processing rate of workstations and transportation time between these can be planned in a way such that there is no need for buffer storage between workstations. Configurations for a transfer line include: straight-line flow, U-flow, circular flow, open field layout (see Figure 2.11). This type of layout is suitable for high-volume production. Although its production rates are unmatched by other types of layout, it is inflexible in the number of products manufactured and, in case of equipment failure, the complete system stops [Tompkins et al., 2010].

Centralised Layout

In a centralised layout, workstations are layout around the area where materials are transported. Materials enter and flow in this area, from workstation to workstation, as demanded by the required operations (see Figure 2.12), the sequence of which can be altered without modifying the layout. In case of equipment malfunction, and depending on the affected equipment, the system might remain partially operational [Tompkins et al., 2010]. Material handling system configurations that support a centralized layout are: uni-directional linear layout, spine layout, loop layout and star layout.

Functional Layout

In a functional layout, resources with the same type of functionality share the same location (see Figure 2.13). In spite of offering the greatest flexibility for high product variety and/or small production vol-

umes, this layout presents inefficient material handling and complex scheduling, that results in poor lead times, poor resource utilization and limited throughput rates. Furthermore, changes in the product mix and/or routings, often require the redesign of the material handling system [Benjaafar et al., 2002]. This layout can be supported by a material handling system configured in ladder layout.

Cellular Layout

Manufacturing and material handling equipment can be grouped into cells, in which case the layout of the machines has to be determined within each cell as well as the layout of the cells. A cellular layout is suitable for manufacturing systems that produce a large number of components and for which manufacturing activities can be decomposed into almost mutually independent cells [Heragu, 2006]. This design approach simplifies workflow and reduces material handling; however, it can be highly inflexible to the introduction of new products, as it is designed for a fixed set of part families. Some variations of this layout try to surpass this limitations with overlapping cells and machine sharing [Benjaafar et al., 2002].

Distributed Layout

In a distributed layout, resources with the same type of functionality can be more or less distributed through the shop floor (see Figure 2.14). Consequently, accessibility to distributed types of resource is increased, as a result of shorter material travel distances, which makes this layout more efficient for a larger set of product volumes and mixes [Benjaafar et al., 2002].

Modular Layout

A modular layout is composed by a network of basic modules, which consist of groups of machines required for subsets of operations in different routings that are arranged into traditional layout configurations, such as: functional, transfer line or cellular, that minimize total flow distances or costs. To support evolution of the product mix and demand changes, modules can be added and eliminated. It is suitable for manufacturing multiple products [Benjaafar et al., 2002].

Reconfigurable Layout

In a reconfigurable layout, it is assumed that resources can be easily relocated, supporting highly variable product demand and product mix. However, production is affected during relocation of resources, and the relocation itself has costs [Benjaafar et al., 2002].

2.6.4 Layout Design With Genetic Algorithms

To and Ho [2002], To [2003] propose a GA for configuring reconfigurable conveyor-components in a flexible assembly line system. Their approach is a messy genetic algorithm in which a chromosome is a complete assembly line of conveyors. Each chromosome is composed of composite genes, which represent conveyor components, and each component gene represents a property of the conveyor, such as moving direction and rotational angle. In crossover, chromosomes exchange sequences of composite genes that may have different lengths, in which case the offspring chromosomes have a different length from what their parents had. Hence, the "messy" part of the algorithm. When mutation takes place in a chromosome, characteristics, which are encoded in binary as genes, are altered. In fitness evaluation, the composite genes in a chromosome are decoded sequentially. The first composite gene represents the conveyor component closest to the loading station and the following represented conveyor components are assigned positions after that, according to the positions determined using geometric equations specific to the type of conveyor considered. The unknowns of these equations are the characteristics represented by genes and the result is a position for the next conveyor component. Each chromosome is evaluated using a set of fitness functions which, one by one, add to the fitness value of the chromosome, which is zero at first.

2.7 Graphs

Graphs are used in this thesis as a means of analysing the manufacturing system's layout. Diestel [2006] gives the following definitions for graph, path and tree, which are the three relevant structures in the context of this thesis:

> "A graph is a pair $G = (V,E)$ of sets such that $E \subseteq [V]^2$; thus, the elements of E are 2-element subsets of V. ... The elements of V are the vertices (or nodes, or points) of the graph G, and the elements of E are its edges (or lines)."

> "A path is a non-empty graph $P = (V,E)$ of the form

$$V = \{x_0, x_1, ..., x_k\} \qquad E = \{x_0x_1, x_1x_2, ..., x_{k-1}x_k\},$$

> where the x_i are all distinct. The vertices x_0 and x_k are linked by P and are called its ends; the vertices $x_1, ..., x_{k-1}$ are the inner vertices of P. The number of edges of a path is its length, and the path of length k is denoted by P^k."

> "A topological spanning tree of G is an arc-connected standard subspace of $|G|$ that contains every vertex and every end but contains no circle."

2.7.1 Integrating Graphs In Custom Applications: JGraphT

JGraphT [Naveh, 2003 - 2005] is a free Java graph library that provides mathematical graph-theory objects and algorithms. It supports various types of graphs, including: directed and undirected graphs, graphs with weighted / unweighted / labeled or any user-defined edges, etc. Graph vertices can be of any objects. Furthermore, it can be integrated with the JGraph [JGraph, 2001 - 2010] library, to enable graph visualization.

The Dijkstra algorithm, as explained in Tanenbaum [2003, pages 353 - 355], and the Kruskal minimum spanning tree algorithm are between the algorithms available in JGraphT.

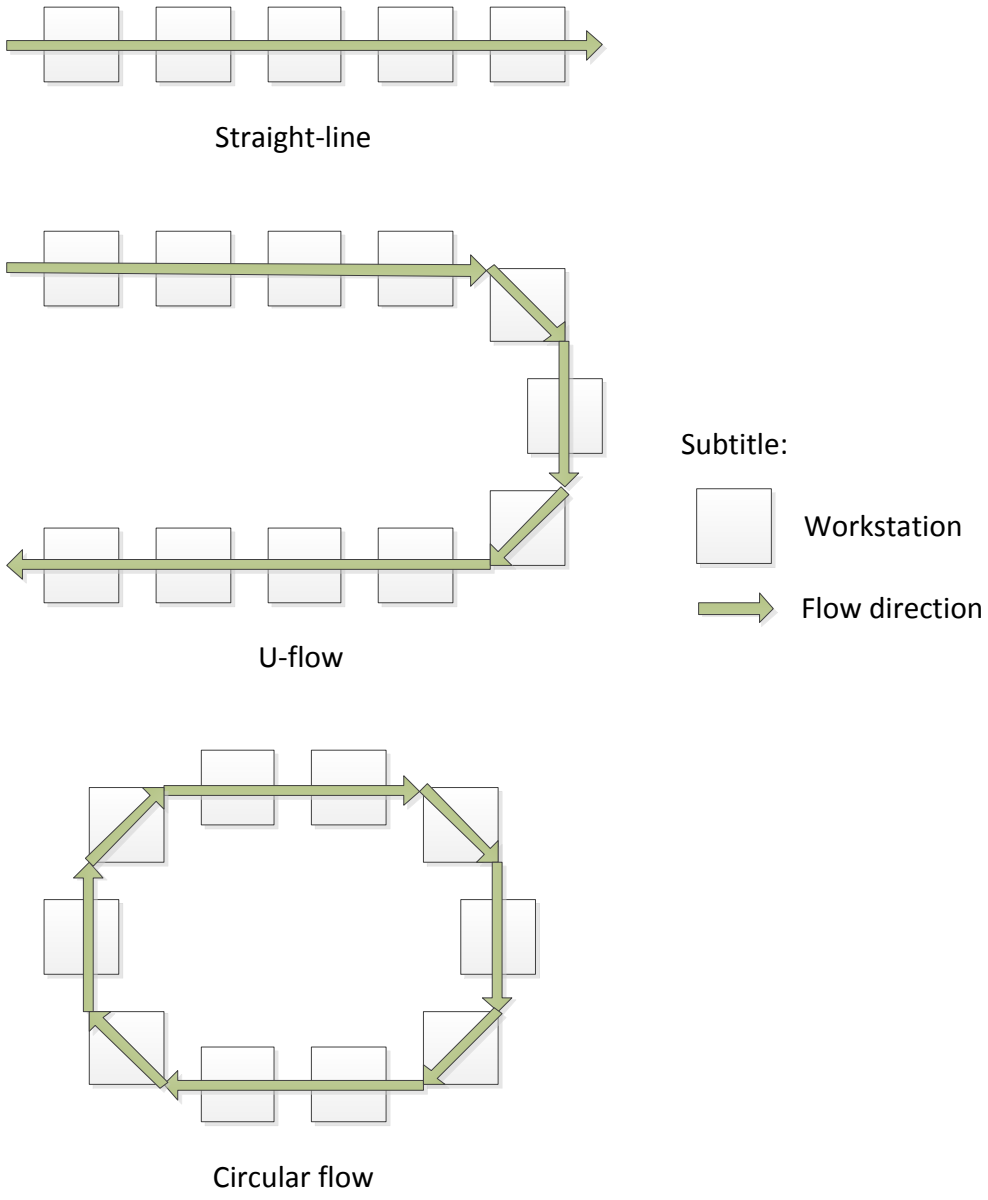

Figure 2.11: Transfer line layouts

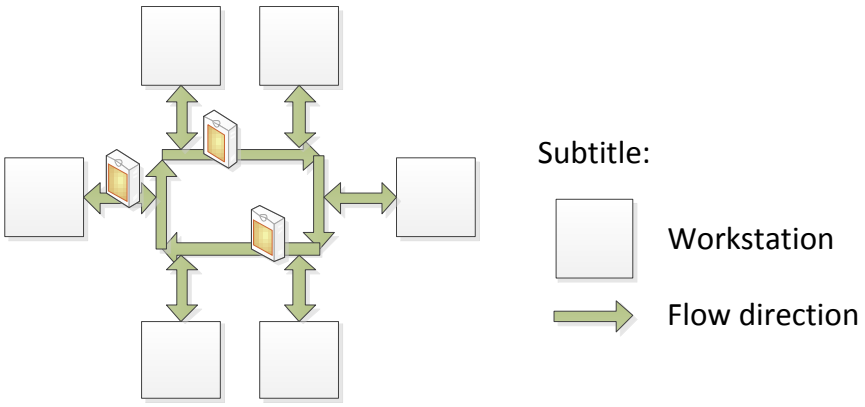

Figure 2.12: Centralised layout

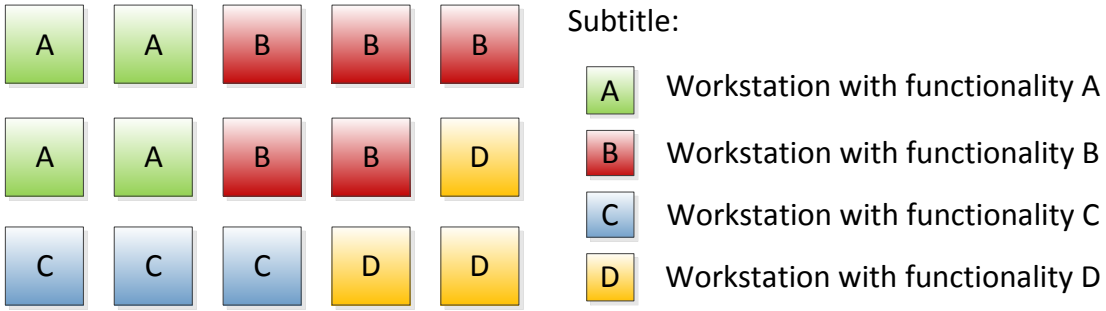

Figure 2.13: Functional layout

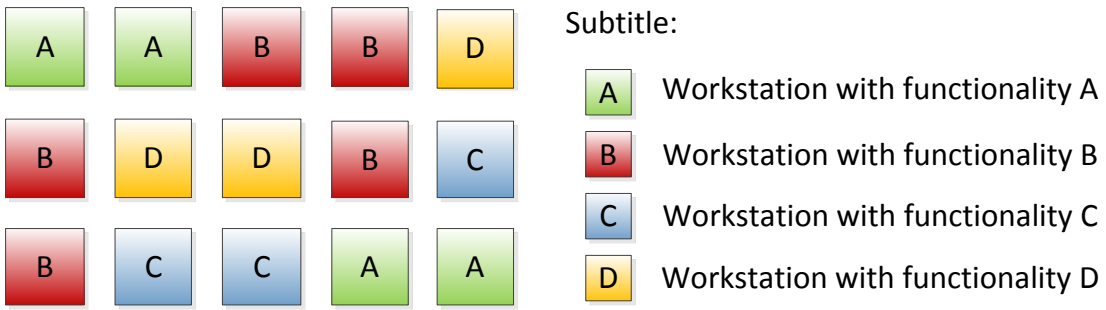

Figure 2.14: Distributed layout

Chapter 3

System Architecture

The purpose of the developed system architecture is to support manufacturing system lay out based on a given product plan. Therefore, it was developed to support design, run-time and re-engineering phases. Modules, in an EAS context, are agents organised in a MAS and exhibit self-* capabilities. Knowledge and process control are distributed through the agents, as different agents have different roles in the manufacturing process, which they perform with a certain degree of autonomy, based on the knowledge they have of themselves, of other agents, and of the process. With this approach, should the need for re-lay out arise during an evolution stage, old modules can be removed and new modules can be added without reprogramming.

This chapter describes and explains the ideas followed and decisions taken during the development of the system architecture in sections with different points of view, focusing on the layout agent, which is the agent capable of performing layout design, and its interactions with other agents.

3.1 Control Approach

In the developed manufacturing system architecture, control results from the interactions between the agents in the MAS. As these interactions evolve, the architecture is able to adapt to different system life cycle phases.

3.1.1 Design Time

The first agent to be launched is the Directory Facilitator (DF), which is followed by resource agents, and then by the layout agent. Resource agents register to the DF and send it their characteristics and capabilities. The layout agent registers to the DF and requests information about the resource agents, which is used to design a layout for the manufacturing system by assigning positions to the resources controlled by resource agents (see Figure 3.1).

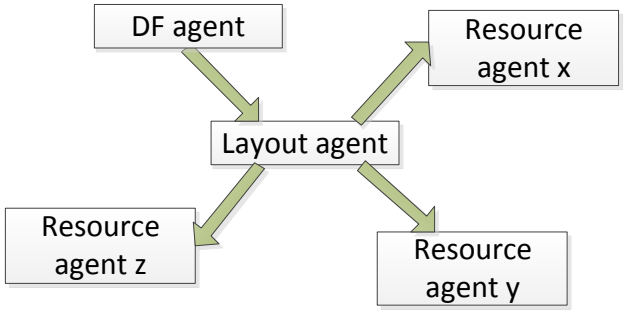

Figure 3.1: Information flow at the beginning of design time

After the positions have been assigned, each resource agent interacts with its neighbour agents, which are the ones that control resources that were assigned positions near its resource, to create coalitions (see Figure 3.2). It is over when it is not possible to create more coalitions. Resource agents update their registry information at the DF.

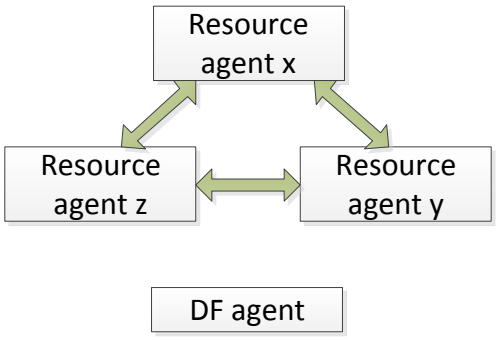

Figure 3.2: Information flow at the end of design time

3.1.2 Run-Time

At this stage, the system agent is launched. It registers the DF and requests information about the resource agents. Furthermore, it positions itself at the top level of an agent hierarchy, issuing control orders from the manufacturing plan. Resource agents are at the bottom layers, receiving control orders from the system agent, distributing these orders among them, and controlling the manufacturing resources (see Figure 3.3).

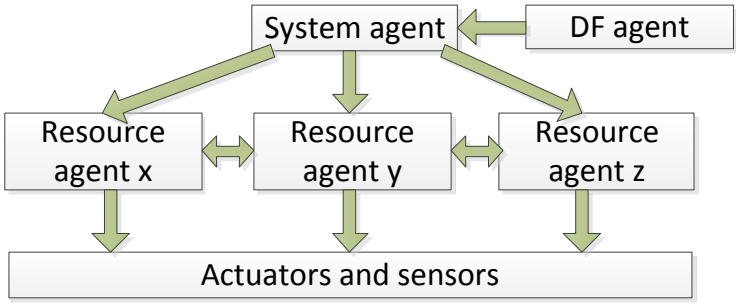

Figure 3.3: Control approach at run-time

3.2 Types of Agents

There are resource agents, a layout agent, a system agent, and a DF agent, in the MAS. They have different knowledge and different behaviours, that change throughout the life cycle of the system.

3.2.1 Resource Agent

A resource agent is initialised with an ontology file containing a description of what type of resource it has to become and a description of the physical device it controls. It contains an ontology model where this information is kept and from where it can be retrieved and used by the agent to create coalitions and to issue control actions to the device hardware. It offers its functionalities, as described in the ontology model, as skills to other agents, which can request their execution. Some resource agents have skills that can only be offered in a joint effort with other resource agents. In order to make these complex skills available, resource agents create coalitions with one-another, that are controlled by one of them.

3.2.2 Layout Agent

A layout agent is initialized with an ontology file containing a product manufacturing plan, which is kept in an ontology model inside the agent. It requests, to the DF, the descriptions of resource agents and keeps them in the ontology model. This knowledge is used, by the agent, to design the layout of the manufacturing system. Based on the layout designed, it assigns a position to each manufacturing resource, by sending a message to the corresponding agent.

3.2.3 System Agent

A system agent is initialized with an ontology file containing a product manufacturing plan, which is kept in an ontology model inside the agent. It requests, to the DF, the descriptions of resource agents and

keeps them in the ontology model. This knowledge is used, by the agent, to coordinate resources and coalitions when the plan is being executed.

3.2.4 Directory Facilitator Agent

The DF accepts registration requests from the layout agent, system agent and resource agents. Registered agents send it information, which it stores in an ontology model, and that other agents might request.

3.3 Agent Knowledge

The system knows what is the product to produce and what resources can be used to produce it through ontology files that are given to the agents when they are initialized, and that contain instances of an ontology that is common to all the agents in the system. This ontology is described in Chapter 4. The product is known through its manufacturing plan. Available resources are known through resource descriptions. The success of layout design is dependent on the quality of this information, as lack of information might cause layout design failure.

Knowledge is stored, reasoned with, and retrieved from an ontology model that exists inside all the agents. The scope and nature of the information contained in the ontology model of an agent depends on the type of agent: the layout agent and the system agent contain information related with the manufacturing plan and with resource agents; and a resource agent contains information about the resource it controls, and about the agents that are part of its coalitions, if that is the case.

Knowledge is provided in OWL, which was chosen because it is a semantic language and the ontologies and instances written in it can be integrated in applications through existing libraries, and easily understood and edited by users.

3.3.1 Manufacturing Plan

The manufacturing plan consists of a sequence of operations with the corresponding arguments (see Figure 3.4). It is supplied, by the user, to the system agent, as a workflow that might include parallel, concurrent and decision dependent operations. Arguments define the conditions by which an operation is to be performed (i.e., length, depth, rotation, velocity, etc.).

Figure 3.4: A manufacturing plan structure

3.3.2 Resource Description

Resource knowledge consists of descriptions of manufacturing resources available to be used in the manufacturing process (i.e., conveyor belts, drillers, robots, cranes, etc.), including their properties and characteristics (i.e., size, weight, etc.), as well as their capabilities and functionalities (i.e., drill, solder, etc.). Capabilities are described as skill templates, which are skills that can only be performed if a set of predefined requirements are met. Capabilities have restrictions that are related to the characteristics of the resources (i.e., a driller with an arm of a certain length can drill only to a certain depth). Functionalities are described as agent skills whose requirements have been fulfilled.

Skills that require cooperation between resource agents are described as complex skills. Their description includes the sequence of interactions that are required for their execution.

Resources have characteristics that are related with the physical connections that have to be established between the resources for the execution of complex skills (i.e., a conveyor belt holds the part while the driller drills a hole in it). The connection between two pieces can be established by overlapping their work volumes. This overlap creates an intersection volume from which one point is chosen as a connection point in the description of the resource. A direction by which this point can be accessed is also included in the description as a connection point orientation.

In Figure 3.5, the conveyor belt contains three connection points: cb:1 and cb:2 as input and output from and to other conveyor belts, and cb:3 in the middle of the belt surface, where it can hold a product part for the driller to work with. This third connection point is the connection point d:1 of the driller.

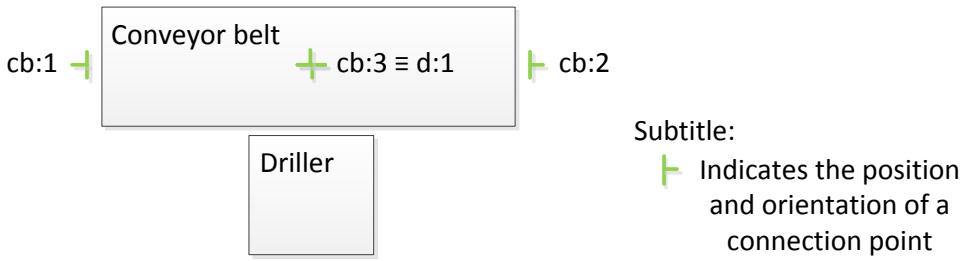

Figure 3.5: Connection points of resources

3.4 Agent Behaviours

The way in which agents behave throughout the manufacturing system life cycle is dependent on the agent behaviours that they are executing inside. Each type of agent has its own set of behaviours, which change throughout each stage of the life cycle.

An initiator and a responder are used for the interaction between an agent and other agents, being

that the initiator behaviour is in the agent that initiated the interaction, and the responder behaviour is in the agents that respond. This interaction follows FIPA communication protocols and uses the FIPA ACL in messages.

3.4.1 Registrations In DF

When they enter the system, the layout agent, system agent and resource agents register in the DF and send it information from their ontology models. The layout agent registers when it is launched. Its layout design capabilities are registered as services. The system agent registers when it is launched. Its product manufacturing plan execution capabilities are registered as services. Resource agents register when they are launched. Their characteristics and skill capabilities are registered as services.

3.4.2 Information Requests To DF

The layout agent and the system agent request information related to the characteristics and capabilities of resource agents, to the DF, through a FIPA request interaction protocol, and insert it into their ontology models.

3.4.3 Layout Design

The layout design behaviour is executed in the layout agent. Its purpose is to design and re-design, when required, the layout of the manufacturing system, for which task it uses a genetic algorithm (see Chapter 5 for details on the developed algorithm). It consists in the selection of candidate agents, algorithm setup, algorithm execution, and reporting of results.

Agent Selection

The layout agent uses the knowledge of the characteristics and capabilities of resource agents for the selection of agents. The agents that can perform the operations required in the manufacturing plan, within the required conditions, are selected to participate in the manufacturing process. This selection is performed by matching, for each manufacturing operation and for each agent, the planned operation conditions with the operation restrictions of the candidate. Although these agents will definitely be in the final solution, all resource agents are selected as candidates to participate in the manufacturing process.

Algorithm Setup

The resources of selected agents are checked for connection points, which are used to initialize the population of chromosomes used by the GA.

Algorithm

Layout design is accomplished by means of a GA. The result of this algorithm is a layout for the manufacturing system, which includes the connections between the resources that were selected to be part of the system. Geometric equations, based on the geometry of each resource, are then used to determine the location and orientation of each one inside the shop floor. However, layout design will provide an incomplete solution if the resources available are not sufficient to perform the manufacturing process required to produce the product.

The GA gives an optimal solution, which might not be the best, within an acceptable length of time. In order to obtain the best solution, all the combinations of connections between resources had to be explored, evaluated and compared, which is a process that increases in length with the increase in connection possibilities. The GA performs a directed search for a solution based on predefined evaluation functions. Different evaluation functions reflect different views of what the best layout should be and lead to different layout solutions.

Reporting Results

The layout agent sends to each resource agent, that was selected to be part of the system, its position and orientation inside the shop floor.

3.4.4 Coalition Creation

A coalition creation behaviour executes inside a resource agent, after it receives a position and orientation. Its purpose is to discover which other resource agents were positioned nearby, that have compatible capabilities and overlapping connection points with this agent, and create coalitions with them.

3.4.5 Workflow Execution

The workflow behaviour exists inside the system agent. It follows the right sequence of skills to request to other agents, from the ones that are described inside the workflow of the product manufacturing plan.

3.4.6 Coalition Coordination

The coalition coordination behaviour exists inside resource agents. It follows the right sequence of skills to request to other agents, from the ones that are described inside the workflow of a complex skill execution plan.

Chapter 4

Supporting Ontologies

Ontologies are used in this thesis to define the product manufacturing plan and the properties, characteristics, capabilities and functionalities of manufacturing resources and of the agents that represent and control them. They are also used to specify a common set of terms that the agents can use to communicate with one-another.

This chapter describes the ontologies developed to support the architecture described in Chapter 3 and the genetic algorithm explained in Chapter 5. Ontologies were written in OWL with the Protégé editor.

4.1 Product Manufacturing Plan

The concepts related with the product manufacturing plan are:

- Product: a product to be manufactured.
- Skill: a functionality required by a product.

The object properties related with the product manufacturing plan are:

- needsSkill(x, y): product x needs skill y.

The datatype properties related with the product manufacturing plan are:

- hasID(x, y): product x has string identification y; skill x has string identification y.

4.2 Resources

The concepts related with the manufacturing resources are:

- ConPoint: a connection point of a resource module.

- Dim: a dimension.
- Module: a resource module.
- Pos: a position.
- Skill: a functionality performed by a resource module.
- ComplexSkill: a functionality that is a result of the execution of a set of functionalities.
- System: a manufacturing system.

The object properties related with the manufacturing resources are:

- compModule(x, y): system x is composed by module y.
- compSkill(x, y): complex skill x is composed by skill y.
- forSkill(x, y): connection point x exists for skill y.
- hasConPoint(x, y): resource module x has connection point y.
- hasDim(x, y): resource module x has dimension y.
- hasPos(x, y): resource module x has position y.
- hasSkill(x, y): resource module x has skill y.

The datatype properties related with the manufacturing resources are:

- hasID(x, y): module x has string identification; connection point x has string identification; product x has string identification y; skill x has string identification y.
- height(x, y): dimension x has a height of y m (double).
- width(x, y): dimension x has a width of y m (double).
- x(x, y): position x has a x coordinate of y m (double).
- y(x, y): position x has a y coordinate of y m (double).
- yaw(x, y): position x has a yaw of y m (double).

These datatype properties correspond to the properties and characteristics of manufacturing equipment available in a set of analysed data sheets and specifications from industrial equipment manufacturers: KUKA, ABB, Montech, and SCHUNK.

Chapter 5

Genetic Algorithm For Layout Design

The genetic algorithm used in this thesis differs from the ones reviewed in Chapter 2 in the way the information is encoded in chromosomes and in the fitness evaluation function.

5.1 Chromosome

A chromosome is composed of multiple composite genes, each one with a number of genes inside. As for the information contained in each of these structures: a chromosome contains encoded information about the layout of the manufacturing system, a composite gene contains encoded information about the connection points of a resource, and a gene contains encoded information about a connection point belonging to the resource (see Figure 5.1). The value of a gene is an identification of a connection point of other resource.

Connection points are selective in the way they connect, which means that, for a connection point of a resource, there is a set of connection points of other resources to which this connection point can establish a connection.

In Figure 5.2, a conveyor belt is connected to a loader through connection point cb:2, to a driller through connection point cb:3, and to an unloader through connection point cb:1.

5.2 Selection Of Parents

Pairs of parent chromosomes are selected through a roulette wheel algorithm.

5.3 Crossover Operator

There is a probability, which can be adjusted, for crossover to happen during reproduction. In crossover, a locus is randomly chosen from one of the parent chromosomes. The group of genes inside the composite

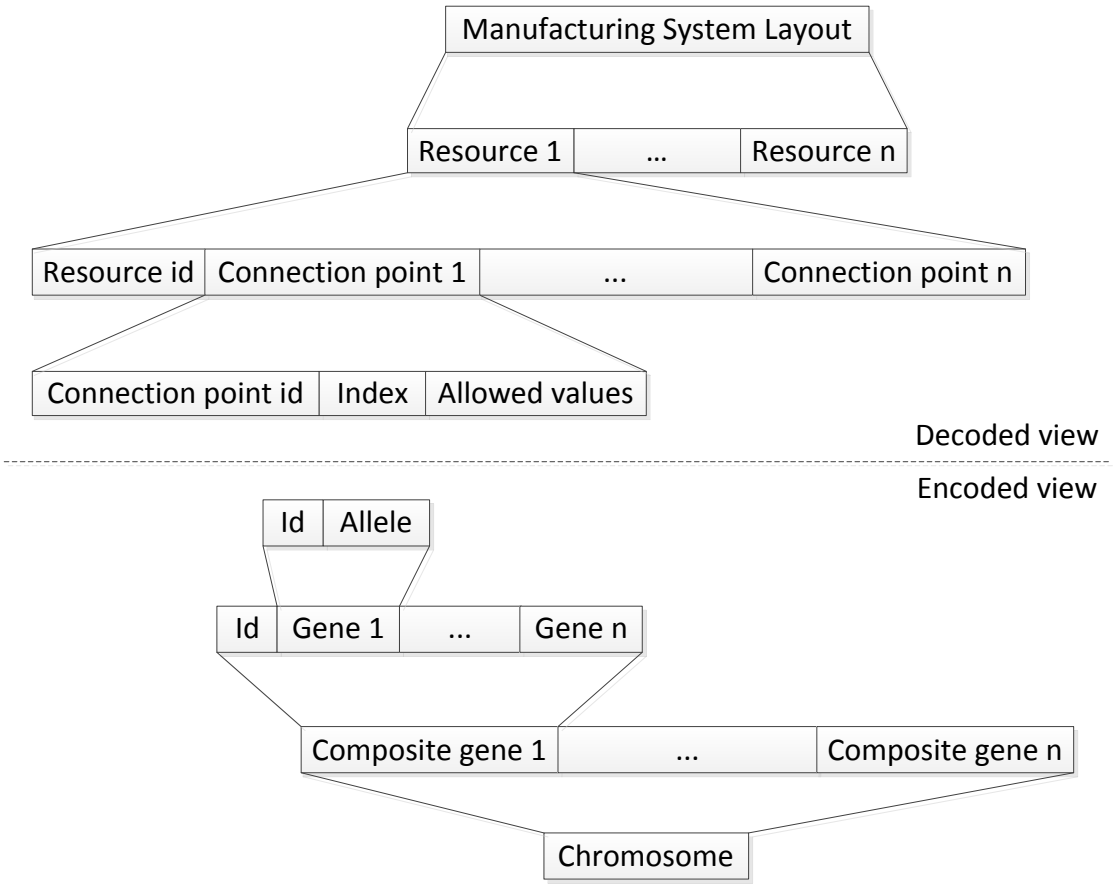

Figure 5.1: Information encoding in a chromosome

gene located at that locus, which represents a group of connections belonging to a resource, changes from one parent chromosome to the other (see Figure 5.3). The composite genes are the same and at the same order in both chromosomes; so, crossover is equal to changing a group of connections at the same time.

The values of genes that were already in a chromosome are changed to match the values of the new genes. This means that, at the end of crossover, the origin connection points take the value of the destiny connections points, and the destiny connection points take the value of the origin connection points. Other genes involved in this process are given a null value.

5.4 Mutation Operator

There is a probability, which can be adjusted, for the composite genes of each chromosome to mutate. In mutation, the value of each gene inside the composite genes that were selected to mutate changes from its current value to other value contained in a set of possible values (see Figure 5.4). The value selected corresponds to a connection point belonging to other resource, from the ones that are compatible with

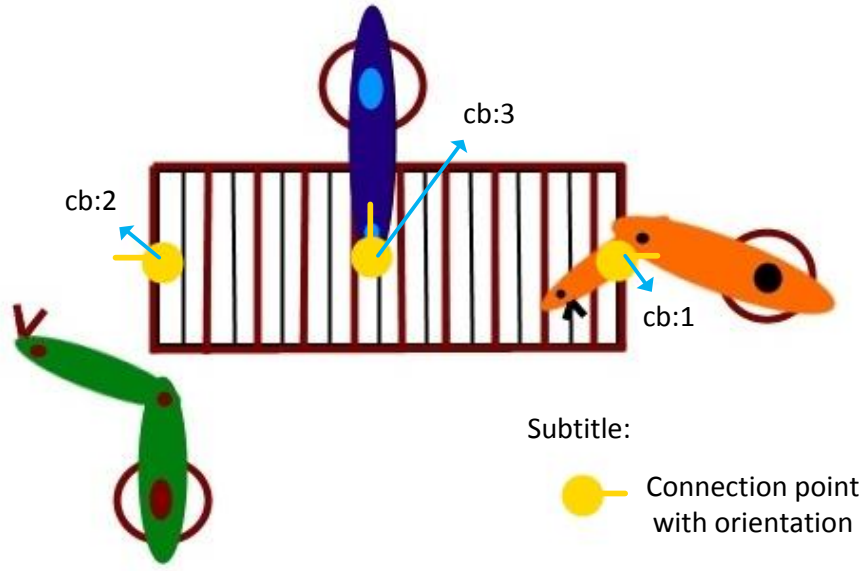

Figure 5.2: Connection points of a conveyor belt

the connection point for which a connection is being established.

At the end of mutation, the origin connection points take the values of the destiny connection points, and the destiny connection points take the values of the origin connection points. Other genes involved in this process are given a null value.

5.5 Fitness Evaluation

The fitness of a chromosome is evaluated, at first, by verifying the existence of paths between the required resources (see Figure 5.5). The fitness value of a solution is inversely proportional to the number of resources through which these paths are established. Then, a layout is designed for the resources that are connected. If at least two resources overlap, the fitness value is reduced to zero, if not, the previous fitness value is maintained and returned.

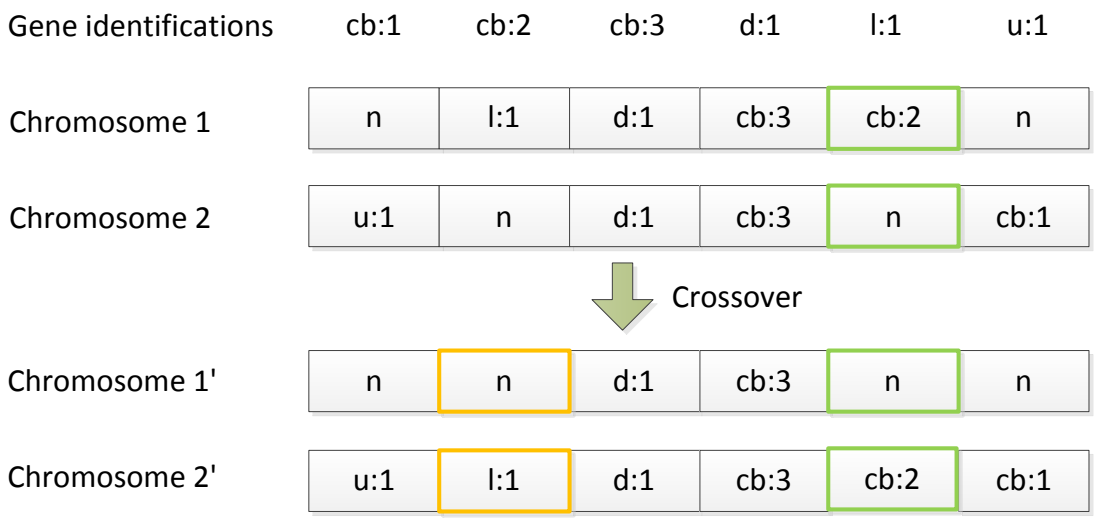

Figure 5.3: Crossover in the layout algorithm

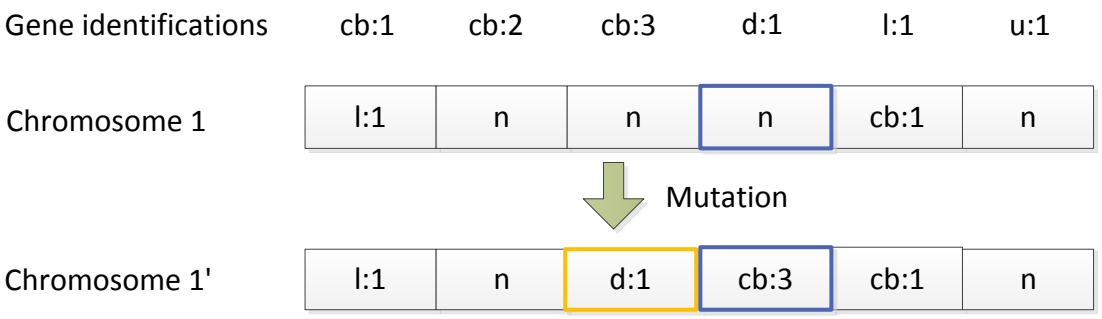

Figure 5.4: Mutation in the layout algorithm

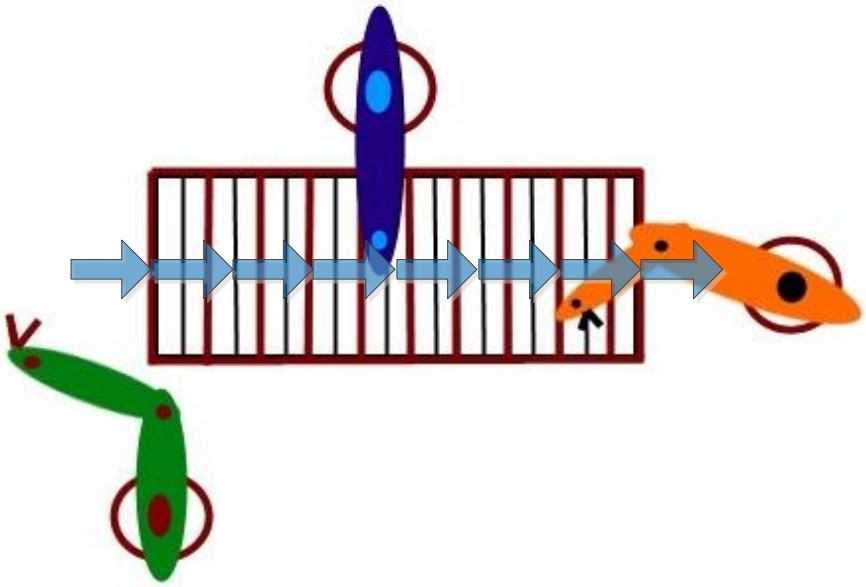

Figure 5.5: A path in a layout

Chapter 6

Implementation

The software application that results from the implementation of the proposed architecture was developed by a team of two master's students in Netbeans Integrated Development Environment (NetBeans) using the Java programming language, being that the part of the implementation performed by the author of this thesis consisted of a layout agent class, a layout design behaviour class, and their supporting classes. The libraries used for agent development, ontology integration, GAs, graph search algorithms and graph drawing are JADE, Jena, JGAP, JGraphT, and JGraph, respectively. The architecture was implemented by extending classes provided by these libraries with new fields and methods.

This chapter describes the process of implementation, the problems, and the solutions.

6.1 Programming In Java

Java technology [Oracle] provides a class-based, object-oriented, platform-independent, multithreaded programming environment, which is why it was chosen over other programming languages. It is the foundation for web and networked services, applications, robotics, and other embedded devices.

6.2 NetBeans

NetBeans [Oracle, 2010] is a development environment that can be used with the Windows Operating System (OS). The NetBeans project consists of an open-source Integrated Development Environment (IDE) and an application platform that enable the creation of web and desktop applications using the Java platform. It was chosen to be used in the implementation for its compiler, debugger, project manager, library manager, subversion support, and GUI editor.

6.3 The Layout Agent

The layout agent and its behaviours were implemented with JADE, as it contains classes for agents, behaviours that implement FIPA agent communication protocols, and messages in the FIPA ACLs.

A layout agent is an instance of the LayoutAgent class, which is an extension of the GuiAgent class. It has a constructor, a setup method and supports GUI events.

6.3.1 Constructor

The constructor of the layout agent class initializes a list where the best layout solutions found by the layout algorithm at each generation are kept.

6.3.2 Setup

At setup, the layout agent shows a GUI through which the user can interact with the agent.

6.3.3 Events

The layout agent supports the events:

- begin layout design - The agent adds the layout behaviour to its running behaviours. The empty list of solutions is passed as an argument.
- show final solution - The agent initializes an index with a value equal to the last index of the, now populated, list of solutions, gets the solution at that index, and calls the graph drawing and the layout drawing methods of the GUI with the graph and layout of the solution as arguments, respectively.
- show first solution - The agent sets the index value to zero, gets the solution at that index, and calls the graph drawing and the layout drawing methods of the GUI with the graph and layout of the solution as arguments, respectively.
- show previous solution - If the index value is higher than zero, the agent decreases the index value, gets the solution at that index, and calls the graph drawing and the layout drawing methods of the GUI with the graph and layout of the solution as arguments, respectively.
- show next solution - If the index value is lower than the last index of the list of solutions, the agent increases the index value, gets the solution at that index, and calls the graph drawing and the layout drawing methods of the GUI with the graph and layout of the solution as arguments, respectively.

6.4 Graphical User Interface Of The Layout Agent

The GUI of the layout agent contains three panels (see Figure 6.1):

- graph panel - where the graph of a solution is drawn.
- layout panel - where the layout of a solution is drawn.
- control panel - where the user can change the default values for the control parameters of the genetic algorithm, begin the layout design process, and change the visible solution.

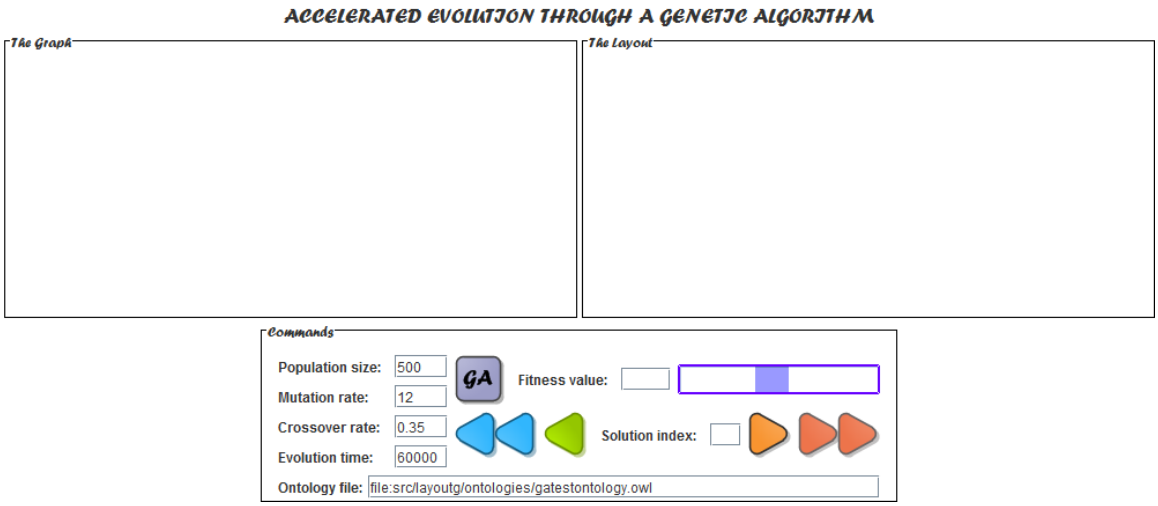

Figure 6.1: Graphical user interface of the layout agent

6.4.1 Control Parameters

Through the GUI, the user can change the parameters: population size, mutation rate, crossover rate, and evolution time, being that the last one is the time limit for the genetic algorithm to reach a solution.

6.4.2 Starting Layout Design

The user can start the layout design process by clicking the layout algorithm button, which posts a begin layout design event on the layout agent. A progress bar shows the time that has passed since the beginning of the algorithm and how much is left. A text field shows the evolution of the fitness of the fittest solution.

6.4.3 Consulting Other Solutions

A solution is shown when the progress bar is filled. This solution is the best solution found in the final generation. It is possible for the user to consult other solutions by using the buttons available in the GUI:

- fast forward - posts a ”show final solution” event on the layout agent.
- forward - posts a ”show next solution event” on the layout agent.
- backwards - posts a ”show previous solution” event on the layout agent.
- fast backwards - posts a ”show first solution” event on the layout agent.

The generation to which the solution belongs and its fitness are visible in two text fields.

6.5 Layout Behaviour

The LayoutBehaviour class extends the SimpleBehaviour class. It has a constructor, an action and a done method.

6.5.1 Constructor

The layout behaviour has a number of states through which it passes before finishing its execution. These are: setup, algorithm, finish and finished. The constructor initializes the behaviour state to setup and gives it access to the layout agent, the fittest chromosome list and the gui panel.

6.5.2 Action

The actions taken by the layout agent are dependent on the state of the layout behaviour when the action method is called, which is specified inside the method:

- setup - The population of chromosomes is initialized with a custom configuration, which is initialized with the control parameters visible at the interface, and a time stamp is initialized with the current time. The GUI is updated and the state is changed to algorithm.
- algorithm - If the time elapsed since the beginning of the algorithm exceeds the time limit, then the state is changed to finish. Otherwise, the population is evolved, the fittest chromosome is saved and the GUI is updated.
- finish - A ”show final solution” is posted on the layout agent, the GUI is updated, and the state is changed to finished.

6.5.3 Done Method

If the state is finished, then the behaviour execution can end.

6.6 Ontology Integration

The ontologies were integrated through Jena, which has classes for ontology models, ontology classes, and properties and their corresponding object and datatype values.

The OntologyMethods class contains a number of methods that are used by the other classes to access the information given in the ontology file, and that is kept in an ontology model inside the class. Information is retrieved through queries in SPARQL Query Language For RDF (SPARQL) [W3C, 2006 - 2007] that are now coded inside the methods for testing purposes, but that can be provided in the ontology file. In that way, if the ontology is changed, the queries can be changed outside the program to match the new information. Each method returns a data structure with the retrieved information.

6.6.1 Model Setup

The readModel method is used to create and initialize the ontology model. The model is initialized to use a OWL rules inference engine.

6.6.2 Retrieving Connection Points

The query used in a method for retrieving the resource, connection point, and corresponding compatible connection point, identifications is:

```
SELECT ?resourceId ?conPointId ?allowedValueId WHERE {
?resource rdf:type ga:Resource .
?resource ga:hasId ?resourceId .
?resource ga:hasConPoint ?conPoint .
?conPoint ga:hasId ?conPointId .
?conPoint ga:forSkill ?skill .
?compSkill ga:compSkill ?skill .
?compSkill ga:compSkill ?otherSkill .
FILTER (?skill != ?otherSkill) .
?allowedValue ga:forSkill ?otherSkill .
?allowedValue ga:hasId ?allowedValueId .
}
```

6.6.3 Retrieving Required Resources

The query used in a method for retrieving the resource identifications is:

```
SELECT ?resourceId WHERE {
?product rdf:type ga:Product .
?product ga:needsSkill ?skill .
?resource ga:hasSkill ?skill .
?resource ga:hasId ?resourceId .
}
```

6.6.4 Retrieving Resource Dimensions

The query used in a method for retrieving the dimensions of the resource corresponding to the given resource identification:

```
SELECT ?resourceWidth ?resourceHeight WHERE {
?resource rdf:type ga:Resource .
?resource ga:hasId GIVEN RESOURCE ID .
?resource ga:hasDim ?resourceDim .
?resourceDim ga:width ?resourceWidth .
?resourceDim ga:height ?resourceHeight .
}
```

6.6.5 Retrieving Connection Point Data

The query used in a method for retrieving the x, y, and yaw, of the connection point corresponding to a given connection point identification:

```
SELECT ?conPointX ?conPointY ?conPointYaw WHERE {
?conPoint rdf:type ga:ConPoint .
?conPoint ga:hasId GIVEN CONNECTION POINT ID .
?conPoint ga:hasPos ?conPointPos .
?conPointPos ga:x ?conPointX .
?conPointPos ga:y ?conPointY .
?conPointPos ga:yaw ?conPointYaw .
}
```

6.7 Configuration

The GA classes were implemented by extending the classes provided by JGAP. This package has a set of classes for chromosomes, genes, genetic operators, and fitness functions.

Through the constructor of the configuration class, the parent selector, mutation operator, crossover operator, and fitness evaluator are set to be used during the execution of the genetic algorithm.

6.8 Encoding Chromosomes And Genes

The genetic algorithm solutions are encoded in the chromosomes by using composite genes that are composed of genes with a certain value.

6.8.1 Chromosomes

Besides the methods available in a chromosome, that are implemented in the extended class, a chromosome has methods to set its corresponding graph and layout. It is composed of composite genes.

In the custom chromosome class, the methods isHandlerFor, perform and clone of the Chromosome class have to be overridden by switching the Chromosome constructor by the custom chromosome constructor.

Constructor

The chromosome class contains three constructors. The first two call the default constructors of the Chromosome class. The third sets the graph and layout values to the values given as parameters.

Chromosome Factory

A chromosome factory was implemented to retrieve the information of the resources available in the ontology into a chromosome data structure. The method to create a chromosome uses the OntologyMethods class to retrieve connection points information, codes the resources as composite genes and connection points as genes. The connection points of other resources that are compatible with the connection point of a resource are kept inside the gene that encodes that connection point.

6.8.2 Composite Genes

Resources are encoded as composite genes, which have an identification equal to the identification of the resource.

The newGeneInternal method of the CompositeGene class has to be overrridden by switching the CompositeGene constructor by the custom composite gene constructor.

Constructor

The constructor of a composite gene sets its identification to the value given as parameter.

6.8.3 Genes

A connection point of a resource is encoded as a gene, which contains an identification equal to the identification of the connection point. This string of identification has to contain, first, the identification of the resource to which the connection point belongs, and then, separated by the colon character, a string of characters that distinguishes this connection point from others of the same resource. This kind of identification was used to enable simpler queries and data structures.

Besides the identification, a gene contains a list of identifications that belong to genes that encode connection points that are compatible with the one encoded by this gene, and an index. The index value indicates which identification on the list is the current value of the gene.

The IndexGene class that was implemented extends the BaseGene class, which is an abstract class, which means that its abstract methods must be implemented, aside from the methods that must be overridden.

The methods newGeneInternal and compareTo are overridden by switching the BaseGene constructor by the IndexGene constructor.

Constructor

A gene has two constructors. The first constructor sets the identification, list of compatible connection points, and index, to the values given as parameters. In the second constructor, the value of the index is set to zero, which is the index of the null value character ”n”. The identification ”n” is added to the list of compatible connection points of all the connection points of all resources at the index zero, by the chromosome factory.

Returning The Value

The internal value of the gene is returned by retrieving the identification at the current index of the list.

Setting The Allele

The allele of an IndexGene is set by changing the value of the index to the value of the identification given as parameter, if that identification is in the list, or to the index value given as parameter.

Setting To A Random Value

When it is necessary to set the value of the gene randomly, the value is set to zero. This is due to the fact that, when the population of chromosomes is initialized in the configuration, it is required that a gene has an index of zero. This means that, when the algorithm starts, there are no connections established.

Gene Mutation

When the gene mutates, its index is set randomly to a value that is higher than zero and lower than the size of the list.

6.9 Crossover And Mutation Operators

The custom classes for crossover and mutation extend the classes CrossoverOperator and MutationOperator, from JGAP, respectively, by overriding the methods doCrossover, in the first, and operate, in the second. The actions performed by these methods are the ones described in Sections 5.3 and 5.4.

6.10 Fitness Evaluation

The fitness function class used in this implementation extends the FitnessFunction class provided by the JGAP package. It has a constructor and an evaluate method.

6.10.1 Constructor

The constructor retrieves the resources required to execute the manufacturing plan by using the OntologyMethods class and keeps them in a list inside the fitness function class.

6.10.2 Fitness Function

At first, for the given chromosome, the fitness is set to zero and its graph is determined by using the GraphMethods class. Then, the evaluate method uses the Dijkstra shortest path algorithm to find the existence of a path between each two resources contained in the list by the order they appear. If there is a path, the function adds a value to the fitness that is equal to the number of required resources divided by the number of resources that are in the path that was found.

$$fitness += \frac{size\ of\ the\ list\ of\ resources}{size\ of\ the\ path} \tag{6.1}$$

The resources that are in the graph, but are not part of any path, are removed from the graph, so that they are not considered when the layout is being determined. At last, a layout that is in agreement with the connections contained in the graph is determined. If at least two resources overlap each other, the chromosome is given a fitness value of zero, otherwise, the fitness value is maintained. Before the fitness is returned, the graph and the layout determined are set in the chromosome.

The graph-theory algorithms used in this implementation are included in JGraphT classes.

6.11 Graph Methods

Three utility graph methods were implemented to facilitate the implementation of the fitness function by creating a graph from a chromosome, creating a layout from a graph, and to retrieve the connection points that are the source and the target of the connection between two resources in an edge of the graph.

6.11.1 Creating A Graph From A Chromosome

The identifications of every composite gene contained in the chromosome is added as a vertex to an undirected graph, and if any gene inside that composite gene has a value that connects it to a gene in other composite gene, an edge is added to the graph with the two identifications of the composite genes as source and target. The sources and edges of the graph are not connection point identifications but are, instead, resource identifications.

6.11.2 Creating A Layout From A Graph

The Kruskal minimum spanning tree algorithm is used to determine the order by which the resources have to be layout. By following this order, aside from the first resource, which is layout without any restriction, the resources are layout in positions relative to other resources that are connected to them and that have already been layout.

A resource is layout relative to other resource by considering the relative coordinates and yaw of the two connection points that connect them and also their dimensions. This information is retrieved from the ontology by using the OntologyMethods class. The result is, for each resource, a position (x and y) and an orientation (yaw).

First, the coordinates of the source connection point are rotated, and then translated, from relative to absolute. Then, the new yaw for the target connection point is calculated.

$$\begin{aligned} new\,target\,connection\,point\,yaw &= source\,resource\,yaw \\ &\quad + source\,connection\,point\,yaw + 180 \end{aligned} \tag{6.2}$$

The rotation that is applied to the target connection point is determined.

$$\begin{aligned} target\,connection\,point\,rotation &= new\,target\,connection\,point\,yaw \\ &\quad - target\,connection\,point\,yaw \end{aligned} \tag{6.3}$$

The target connection point is rotated and, finally, the target resource point is equal to the source connection point, in absolute coordinates, when translated in a distance equal to the difference between the

rotated target connection point and the origin, in x and y. Its yaw is equal to the target connection point rotation.

6.11.3 Determination Of The Source And Target Connection Points In A Resource Connection Edge

To determine by which connection points two resources are connected, all the genes of the composite gene which has the source identification, given as a parameter, are verified, and the one with a value that contains the target identification, also given as a parameter, is the source connection point. Its value is the target connection point.

6.12 Transform Methods

The methods presented in this section were implemented to facilitate the application of geometric transforms to the coordinates and orientation of the resources and connection points.

6.12.1 Rotation

The result of a rotation is calculated using a transform matrix.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} cos(rotation) & -sin(rotation) \\ sin(rotation) & cos(rotation) \end{bmatrix} \begin{bmatrix} x0 \\ y0 \end{bmatrix} \tag{6.4}$$

6.12.2 Translation

The result of a translation is calculated by adding the value of the translation in x and y to the point coordinates.

$$x = x0 + translation\ in\ x \tag{6.5}$$

$$y = y0 + translation\ in\ y \tag{6.6}$$

6.12.3 Inversion

An inversion of the point is obtained by negating the x and y coordinates of the point.

$$x = -x0 \tag{6.7}$$

$$y = -y0 \tag{6.8}$$

6.12.4 Rectangular Area And Areas Of Any Shape

This implementation uses rectangular areas for the shapes of resources, for which it uses the Rectangle2D.Double class, and, consequently, the dimension of a resource is described in the ontology in terms of width and height. However, for determining if two resources intersect, the algorithm has to deal with areas of undefined shapes. For that purpose, instances of the Rectangle2D.Double class are converted to instances of the Area class.

6.13 Decoding

Other classes were implemented to keep the data of connection points (x, y, yaw), edges (source, target), and resources (x, y, yaw, width and height). These classes have set and get methods for the data they keep.

Chapter 7

Validation And Test Cases

In this chapter, the tests performed to validate the implementation through the several stages of its implementation are described.

The test cases were given names based on: the number of resources described in the given ontology, that is the number before the letter ”R”, which stands for ”resource”; the number of resources required directly by the manufacturing plan, that is the number before the letter ”P”, which stands for ”plan”; and the number of resources required for material handling and transport purposes, that is the number before the letter ”T”, which stands for ”transport”.

7.1 Test 4R3P1T

The test 4R3P1T was repeated several times during the development of the implementation to assure that the results given were the ones expected.

The ontology file given to the algorithm contained four manufacturing resources with skills, dimensions and contact points, a product manufacturing plan with a list of required operations, and descriptions of complex skills.

Three of the given resources are required directly by the manufacturing plan and one is required for transporting the product parts between these resources.

Every time this test was performed, the algorithm was expected to select the four resources to be part of the manufacturing system. Three of them were to be selected based on the manufacturing plan description. The third was to be selected during the establishment of connections between the resources, due to the fact that it was the only resource available to connect the other three in the required order, according to the descriptions of complex skills and connection points. The algorithm was also expected to design the layout, based on the established connections, and to present a drawing of the same. After each test, the algorithm was tunned to optimize its performance.

With the current implementation, population of 500, mutation rate of 12, and crossover rate of 0.35, the algorithm presents the expected result in less than one second and one generation of solutions (see Figure 7.1).

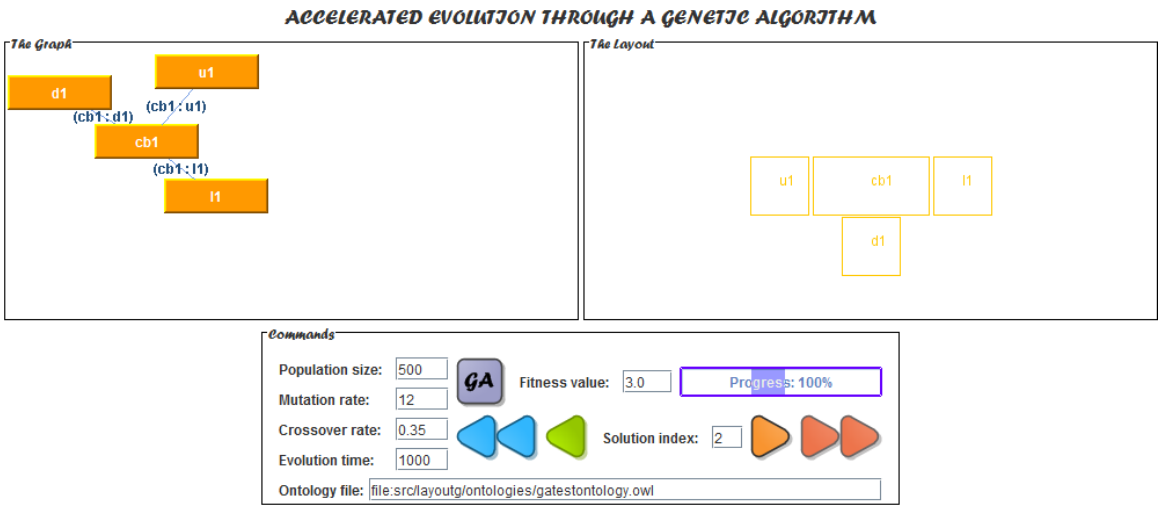

Figure 7.1: Solution presented by the layout algorithm for the test 4R3P1T

7.2 Test 6R4P2T

The test 6R4P2T is an improvement of the test 4R3P1T. Its purpose was to see how the layout algorithm reacted to the introduction of two more manufacturing resources.

The ontology file given to the algorithm contained six manufacturing resources with skills, dimensions and contact points, a product manufacturing plan with a list of required operations, and descriptions of complex skills.

Four of the given resources are required directly by the manufacturing plan and two are required for transporting the product parts between these resources.

The algorithm was expected to select the six resources to be part of the manufacturing system.

With the current implementation, population of 500, mutation rate of 12, and crossover rate of 0.35, the algorithm presents the expected result in less than five seconds and five generations of solutions (see Figure 7.2).

7.3 Test 5R3P1T

The purpose of the test 5R3P1T was to verify that the algorithm was selecting the shortest path possible, and consequently, the fewer resources possible.

The ontology file given to the algorithm contained five manufacturing resources with skills, dimensions and contact points, a product manufacturing plan with a list of required operations, and descriptions of complex skills.

Three of the given resources are required directly by the manufacturing plan, one is required for transporting the product parts between these resources, and one has connection points that are compatible with the connection points of these resources.

The algorithm performed as expected, by selecting the first four resources to be part of the manufacturing system and ignoring the other one.

With the current implementation, population of 500, mutation rate of 12, and crossover rate of 0.35, the algorithm presents the expected result in less than one second and two generations of solutions (see Figure 7.3).

7.4 Test 9R3P2T

Test 9R3P2T is a mix of the tests in Sections 7.1, 7.2 and 7.3. Its purpose was to see how the layout algorithm reacted to the increase of manufacturing resources, being that some of them are identical.

The ontology file given to the algorithm contained nine manufacturing resources with skills, dimensions and contact points, a product manufacturing plan with a list of required operations, and descriptions of complex skills.

Five of the given resources are required directly by the manufacturing plan, being that three of them are equal. The other four are capable of transporting the product parts, but only two of them are required.

The algorithm selected the five resources that offered the skills required by the manufacturing plan and two transporters to connect them. The rest of the transporters were ignored.

With the current implementation, population of 500, mutation rate of 12, and crossover rate of 0.35, the algorithm presents the expected result in less than 30 seconds (see Figure 7.4).

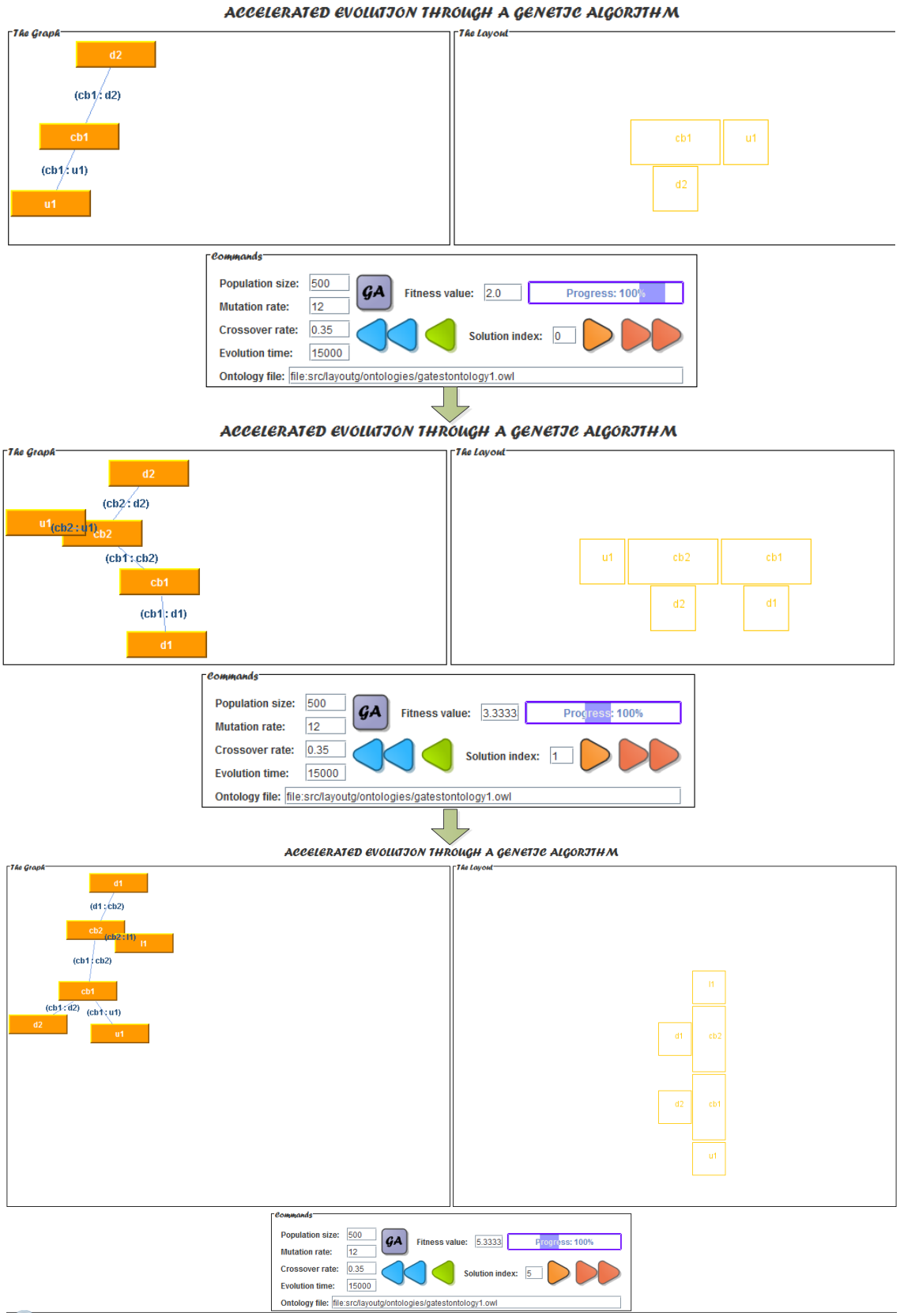

Figure 7.2: Sequence of solutions presented by the layout algorithm for the test 6R4P2T

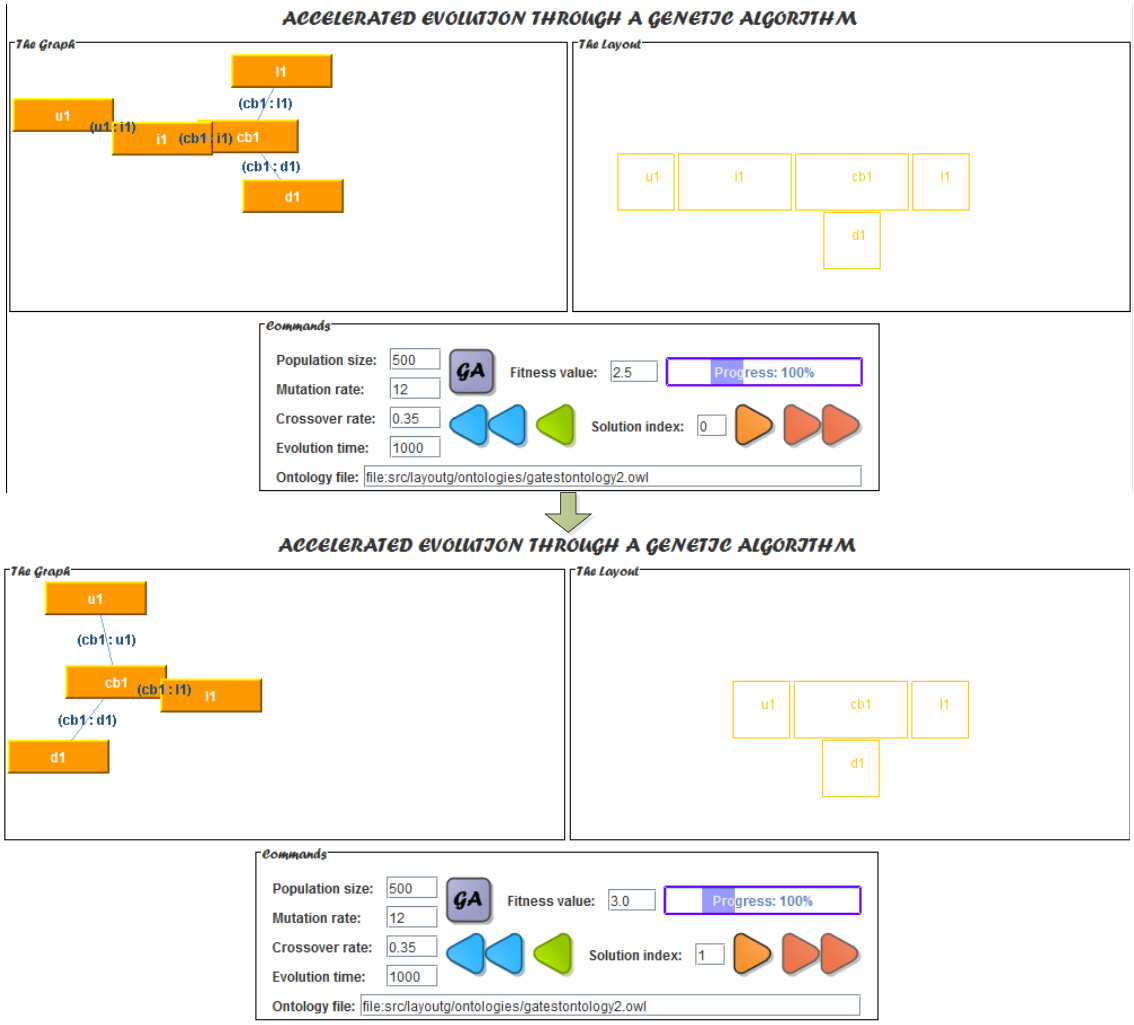

Figure 7.3: Sequence of solutions presented by the layout algorithm for the test 5R3P1T

Figure 7.4: Solution presented by the layout algorithm for the test 9R3P2T

Chapter 8

Conclusions

In Chapter 2, the state of the art on manufacturing systems' paradigms and control approaches was reviewed, related work was acknowledged, and supporting concepts such as agents, ontologies, genetic algorithms and graphs were explained, to set a solid ground for the upcoming chapters.

Chapters 3 through 7 validate the hypothesis formulated in Section 1.3 and fulfill the objectives proposed in 1.4. It was possible to demonstrate the creation of a manufacturing system's layout with the developed algorithm.

Chapter 3 contains a description of an agent architecture that integrates agent technology, ontologies (described in detail in Chapter 4), genetic algorithms (described in detail in Chapter 5) and graph-theory algorithms, which is implemented in Chapter 6 and validated in Chapter 7. The implemented agent can be integrated in a MAS that controls an EAS to provide a new SO-capability. This integration requires the further implementation of specific communication behaviours.

The developed approach is dependent on the quantity and quality of the knowledge it contains. This knowledge is distributed by the agents in the system in a way that every agent has access to the information it requires to function properly. A part of this knowledge, which consists of the manufacturing plan and resource descriptions, is given by the user to the agents when they are initialised. The other part, which is related to positions, orientations and coalitions, is generated by the agents themselves and shared among them, sometimes by using the DF agent and others directly. The reason why the DF agent keeps all the characteristics and capabilities of the resource agents is so it can ”feed” the ”knowledge hungry” agents such as the layout agent and the system agent, who would have, otherwise, to request this information to the resource agents in the system individually, which would involve the trading of a higher amount of messages, proportional to the number of resource agents.

Control is also distributed by the agents in the system, which takes complexity from the design and implementation of each individual agent and puts it in the orchestration of the interactions between all of them. Moreover, functionalities can be easily added and removed from the system by adding

and removing the corresponding agents, which is the situation of layout design and the layout agent. Likewise, a resource agent, for instance, is a generic agent that can become any type of resource that is described in the ontology file that it receives when it is launched in the system. When it is removed, the functionalities of the equipment it represents are no longer offered to the system. Agent technology was chosen because an agent can be designed to keep knowledge inside it, to run the desired algorithms, and to interact with other agents, that is what was required.

The layout agent executes an algorithm based on a biologically inspired genetic algorithm to design the layout of a manufacturing system. A genetic algorithm was chosen because it allows the directed search for a solution, which is, in this situation, a layout that is fit to manufacture the desired product, based on the resources required to execute the activities described in the manufacturing plan. In the current implementation, the algorithm stops searching for a solution when time limit predefined by the user ends. The result might be a complete or incomplete solution, that might be useful either way. The discovery of a good solution is dependent on the descriptions provided in the ontology file, through the queries that retrieve their content, on its control parameters, and on the fitness function. As every case is a different case, it might be useful to change one of these items to accelerate the discovery of a solution, or even to obtain a better one. This is possible only in the case of the descriptions in the ontology and the control parameters, as they are the only ones that are accessible to the designer.

The encoding of information in the chromosomes of the genetic algorithm is accomplished in a generic manner, by which any manufacturing system that is described based on the connections between its manufacturing resources can be encoded. The link between two composite genes in a chromosome is the value of two or more genes. In other words, two resources are connected through their connection points. This enables the appearance of solutions with loops that correspond to a transfer line with a circular flow or a centralised loop layout, which is another difference between this and other existing approaches.

A genetic algorithm can sometimes become stuck in a local minimum solution, which can be a problem if that solution does not have the required resources or connections. As it happens with other genetic algorithms, this approach relies on the mutation operator to avoid this problem.

In the test cases, the scalability of this solution was studied for a maximum of nine manufacturing resources, for which the algorithm took less than 30 seconds to find a good solution. However, this was still a small test. The approach needs to be tested with a higher number of resources, for which a real case could be used.

The solution found is presented in a connection graph and in a two dimensional layout map, which are easily understandable by the designer. The first indicates which connections have to be established between which manufacturing resources, and the second indicates the position and orientation of each of

these resources. It is a task of the designer to move the equipment to the locations defined in the solution.

An article describing early work on this project was accepted in the IEEE International Symposium on Industrial Electronics 2010, held in Bari, Italy, in July 2010 (see Reference [Frei et al., 2010]).

8.1 Future Work

From the conclusions in Section 1.4, although the specified architecture and its implementation validate the hypothesis formulated in Section 1.3 and fulfill the objectives proposed in Section 1.4, future work on the following topics will improve the developed approach:

- The supporting ontology specification should be extended to support the description of more resources and their characteristics.
- The queries should be given as an input to the algorithm, as they are dependent on the ontology specification.
- The control parameters of the genetic algorithm should be tunned more precisely so that the layout can be design in a shorter time.
- The fitness function should be editable outside the program, so that it might be changed without changing the code. This could be done by using a script language.
- The result of the layout design could be a 3D map instead of 2D, so that the designer might have a better perception of where and how to position the resources.

Bibliography

ABB. Abb em portugal, 2011. URL http://www.abb.pt/.

J. L. Austin. *How To Do Things With Words*. Harvard University Press, 2nd edition, 1975.

O. Babaoglu and editors Shrobe, H. Fourth ieee international conference on self-adaptive and self-organizing systems, budapest, hungary, September 27-October 1 2010.

J. Barata. *Coalition Based Approach For Shop Floor Agility*. PhD thesis, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, 2005.

J. Barata and M. Onori. Evolvable assembly and exploiting emergent behaviour. In *Industrial Electronics, 2006 IEEE International Symposium on*, volume 4, pages 3353 –3360, jul. 2006. doi: 10.1109/ISIE.2006.296004.

S. Benjaafar, S. S. Heragu, and S. A. Irani. Next generation factory layouts: Research challenges and recent progress. *Interfaces*, 32(6):58 – 76, 2002. URL http://www.jstor.org/stable/20141207.

R. A. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEE Transactions on*, 2 Issue:1:14 – 23, Mar 1986. ISSN 0882-4967. doi: 10.1109/JRA.1986.1087032.

R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139 – 159, 1991a. ISSN 0004-3702. doi: DOI:10.1016/0004-3702(91)90053-M. URL http://www.sciencedirect.com/science/article/B6TYF-47YRKJD-4D/2/0161f0099e8b4e126a78033b6712d6e1.

R. A. Brooks. New approaches to robotics. *Science*, 253(5025):1227 – 1232, 13 September 1991b. doi: 10.1126/science.253.5025.1227.

R. A. Brooks. Itelligence without reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569 – 595, Sydney, Australia, 1991c.

L. M. Camarinha-Matos, L. S. Lopes, and J. Barata. Integration and learning in supervision of flexible assembly systems. *Robotics and Automation, IEE Transactions on*, 12 Issue: 2:202 – 219, April 1996. ISSN 1042-296X. doi: 10.1109/70.488941.

C. Darwin. *The Origin of Species*. Cricket House Books LLC, 2010.

W. H. Davidow and M. S. Malone. *The Virtual Corporation: structuring and revitalizing the corporation for the 21st century*. HarperBusiness, New York, 1993.

G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi. A generic framework for the engineering of self-adaptive and self-organising systems. In Kirstie Bellman, Michael G. Hinchey, Christian Müller-Schloer, Hartmut Schmeck, and Rolf Würtz, editors, *Organic Computing - Controlled Self-organization*, number 08141 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

I. Dickinson. Jena ontology api, 2009. URL http://jena.sourceforge.net/ontology/index.html.

R. Diestel. *Graph Theory*. Springer, 3rd edition, 2006.

D. M. Dilts, N. P. Boyd, and H. H. Whorms. The evolution of control architectures for automated manufacturing systems. *Journal of Manufacturing Systems*, 10(1):79–93, 1991. ISSN 0278-6125. doi: DOI:10.1016/0278-6125(91)90049-8. URL http://www.sciencedirect.com/science/article/B6VJD-47XF485-21/2/d5bba8967507b5c1332e95c3d5f9b984.

EUPASS. Eupass : Evolvable ultra-precision assembly systems. URL http://cordis.europa.eu/fetch?CALLER=PROJ_ICT&ACTION=D&CAT=PROJ&RCN=75342.

FIPA. Fipa contract net interaction protocol specification, 1996 - 2002a. URL http://www.fipa.org/specs/fipa00029/SC00029H.html. (19 October 2010).

FIPA. Fipa acl message structure specification, 1996 - 2002b. URL http://www.fipa.org/specs/fipa00061/SC00061G.html. (19 October 2010).

FIPA. Fipa request interaction protocol specification, 1996 - 2002c. URL http://www.fipa.org/specs/fipa00026/SC00026H.html. (19 October 2010).

FIPA. Fipa subscribe interaction protocol specification, 1996 - 2002d. URL http://www.fipa.org/specs/fipa00035/SC00035H.html. (19 October 2010).

'flexibility'. Merriam-webster online dictionary, 2010. URL http://www.merriam-webster.com/dictionary/flexibility. (23 September 2010).

R. Frei. *Self-Organisation in Evolvable Assembly Systems*. PhD thesis, Universidade Nova de Lisboa, Faculdade de Cincias e Tecnologia, August 4 2010.

R. Frei and J. Barata. Embodied intelligence to turn evolvable assembly systems reality. In Américo Azevedo, editor, *Innovation in Manufacturing Networks*, volume 266 of *IFIP International Federation for Information Processing*, pages 269–278. Springer Boston, 2008. URL http://dx.doi.org/10.1007/978-0-387-09492-2_29. 10.1007/978-0-387-09492-2_29.

R. Frei, G. Di Marzo Serugendo, and J. Barata. Designing self-organization for evolvable assembly systems. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, pages 97 –106, oct. 2008. doi: 10.1109/SASO.2008.20.

R. Frei, B. Ferreira, G. Di Marzo Serugendo, and J. Barata. An architecture for self-managing evolvable assembly systems. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 2707 –2712, oct. 2009a. doi: 10.1109/ICSMC.2009.5346137.

R. Frei, N. Pereira, J. Belo, J. Barata, and G. Di Marzo Serugendo. Self-awareness in evolvable assembly systems. Technical report, BBKCS-09-07, School of Computer Science and Information Systems, Birbeck College, London, UK, 2009b.

R. Frei, N. Pereira, J. Belo, J. Barata, and G. Di Marzo Serugendo. Implementing self-organisation and self-management in evolvable assembly systems. In *IEEE Int. Symp. on Industrial Electronics (ISIE)*, page 3527 3532, Bari, Italy, 2010.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. ISBN 0201157675.

S. L. Goldman, R. N. Nagel, and K. Preiss. *Agile competitors and virtual organizations: strategies for enriching the costumer*. Van Nostrand Reinhold, New York, 1995.

M. P. Groover. *Automation, production systems, and computer-integrated manufacturing*. Prentice Hall, 3rd edition, 2007.

S. S. Heragu. *Facilities Design*. iUniverse, 2nd edition, 2006.

S. S. Heragu and A. Kusiak. Expert systems in manufacturing design. *IEE Transactions on Systems, Man and Cybernetics*, SMC-17(6):898 – 912, 1987.

S. S. Heragu and A. Kusiak. Machine layout problem in flexible manufacturing systems. *Operations Research*, 36(2):258 – 268, 1988.

Jade - Telecom Italia Lab. Jade - java agent development framework, 2010. URL http://jade.tilab.com/.

Jena. Jena - a semantic web framework for java. URL http://jena.sourceforge.net/.

JGraph. Jgraph, 2001 - 2010. URL http://www.jgraph.com/.

KUKA. Kuka robotics, 2010. URL http://www.kuka-robotics.com.

P. J. P. Leitão. *An Agile and Adaptative Holonic Architecture for Manufacturing Control*. PhD thesis, Faculty of Engineering of University of Porto, Porto, January 2004.

B. Maskell. The age of agile manufacturing. *Supply Chain Management: An International Journal*, 6 (1):5–11, 2001.

B. McBride, D. Boothby, and C. Dollin. An introduction to rdf and the jena rdf api, 2009. URL http://jena.sourceforge.net/tutorial/RDF_API/index.html.

K. Meffert and N. Rotstan. Jgap - java genetic algorithms package, 2002 - 2010. URL http://jgap.sourceforge.net/.

'monopoly'. Merriam-webster online dictionary, 2010. URL http://www.merriam-webster.com/dictionary/monopoly. (20 September 2010).

Montech. Montech, 2011. URL http://www.montech.com/.

B. Naveh, 2003 - 2005. URL http://www.jgrapht.org/.

A. M. C. Nunes, M. B. M. Vargas, and M. J. G. Lobato. *História e Geografia de Portugal - 5o Ano*. Lisboa: O Livro, 1st edition, 1995.

M. Onori, H. Alsterman, and J. Barata. An architecture development approach for evolvable assembly systems. In *Assembly and Task Planning: From Nano to Macro Assembly and Manufacturing, 2005. (ISATP 2005). The 6th IEEE International Symposium on*, pages 19 –24, jul. 2005. doi: 10.1109/ISATP.2005.1511444.

Oracle. About java. URL http://www.oracle.com/us/technologies/java/index.html. (15 November 2010).

Oracle. Netbeans ide, 2010. URL http://netbeans.org/.

OWL Working Group. Owl recommendation, 2004. URL http://www.w3.org/TR/owl-features/.

M. Rajasekharan, B. A. Peters, and T. Yang. A genetic algorithm for facility layout design in flexible manufacturing systems. *International Journal of Production Research*, 36(1):95–110, January 1998.

RDF Working Group. Rdf standards, 2009. URL http://www.w3.org/standards/techs/rdf#w3c_all.

T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohm. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2):209 – 238, 1999. ISSN 0004-3702. doi: DOI:10.1016/S0004-3702(99)00036-3. URL http://www.sciencedirect.com/science/article/B6TYF-3X7VMJC-7/2/b1224a6ab9375f4634384eeb57c2f2b3.

SCHUNK. Schunk, 2011. URL http://www.schunk.com.

J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1997.

W. Shen, M. Onori, J. Barata, and R. Frei. Evolvable assembly systems basic principles. In *Information Technology For Balanced Manufacturing Systems*, volume 220 of *IFIP International Federation for Information Processing*, pages 317–328. Springer Boston, 2006. URL http://dx.doi.org/10.1007/978-0-387-36594-7_34. 10.1007/978-0-387-36594-7_34.

Stanford Center for Biomedical Informatics Research. Protégé website, 2010. URL http://protege.stanford.edu/.

A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, 4th edition, 2003.

T. K. W. To. An evolutionary approach for the reconfiguration of an assembly-line system. Master's thesis, Department of Manufacturing Engineering and Engineering Management of the City University of Hong Kong, Hong Kong, February 2003.

T. K. W. To and J. K. L. Ho. A genetic algorithm for configuring reconfigurable conveyor-components in a flexible assembly line system. In *Manufacturing Complexity Network Conference April 2002. Second International Conference on*, Downing College, Cambridge, UK, 9 - 10 April 2002. Institute for Manufacturing.

J. A. Tompkins, J. A. White, and Y. A. Bozer. *Facilities Planning*. John Wiley and Sons, 4th edition, 2010.

D. M. Upton. A flexible structure for computer-controlled manufacturing systems. *Manufacturing review*, 5(1):58–72, 1992.

W3C. Sparql query language for rdf, 2006 - 2007. URL http://www.w3.org/TR/rdf-sparql-query/.

P. H. Winston and K. A. Prendergast. *The AI business: the commercial uses of artificial intelligence*. MIT Press, 1984.

M. Wooldridge. *An Introduction To MultiAgent Systems*. John Wiley and Sons, 2nd edition, 2009.